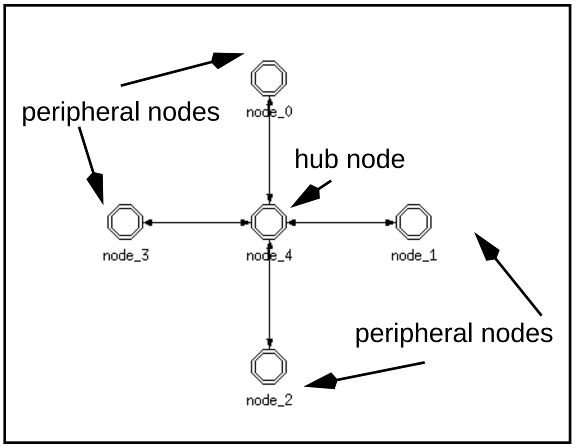
Chapter Pst

Packet Switching Tutorial

Pst.1 Defining the Task

The task in this chapter is to use OPNET to simulate the behavior of a packet switching network consisting of nodes connected by point-to-point links in a star topology. Packets shall travel from source to destination via a central, or hub, packet switching node which has point-to-point connections with each peripheral node. The initial configuration shall contain four peripheral nodes and one hub node as shown below.

Star Topology

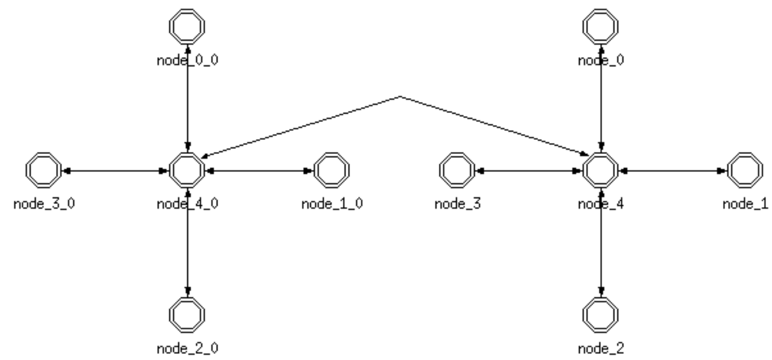


The hub node shall have the single responsibility of relaying packets to the proper destination. The hub node will not initiate any message traffic on its own. Peripheral nodes will generate traffic randomly according to an exponential distribution for interarrival times, and each packet shall contain a random destination address.

Performance shall be measured by the end-to-end delay experienced by packets. End-to-end delay shall be calculated as the number of seconds from the time a packet is created to the time it is received by the destination node. Packet sizes and link data rates shall be constant, so the end-to-end delay will be constant except when packets experience queuing delays in transmitters. Queuing delays occur when packets are sent to a transmitter which is already busy transmitting a packet. Queuing delays can occur in both peripheral nodes and in the hub node since both node types have transmitters.

After analyzing this configuration, the model shall be enhanced with the addition of a second star network. The two star networks will be connected by a point-to-point link between the hub nodes in each star. Each peripheral node will have a unique address. The hub nodes will have the ability to determine if a packet's destination address is within the scope of the local star, or if the packet must be sent to the other hub node. The end-to-end delay for packets traversing the inter-net path will be greater since a third link is involved. This third link delay is also compounded by the fact that greater congestion at the inter-net transmitters results in more queuing delays. The enhanced configuration is shown below.

Enhanced Configuration

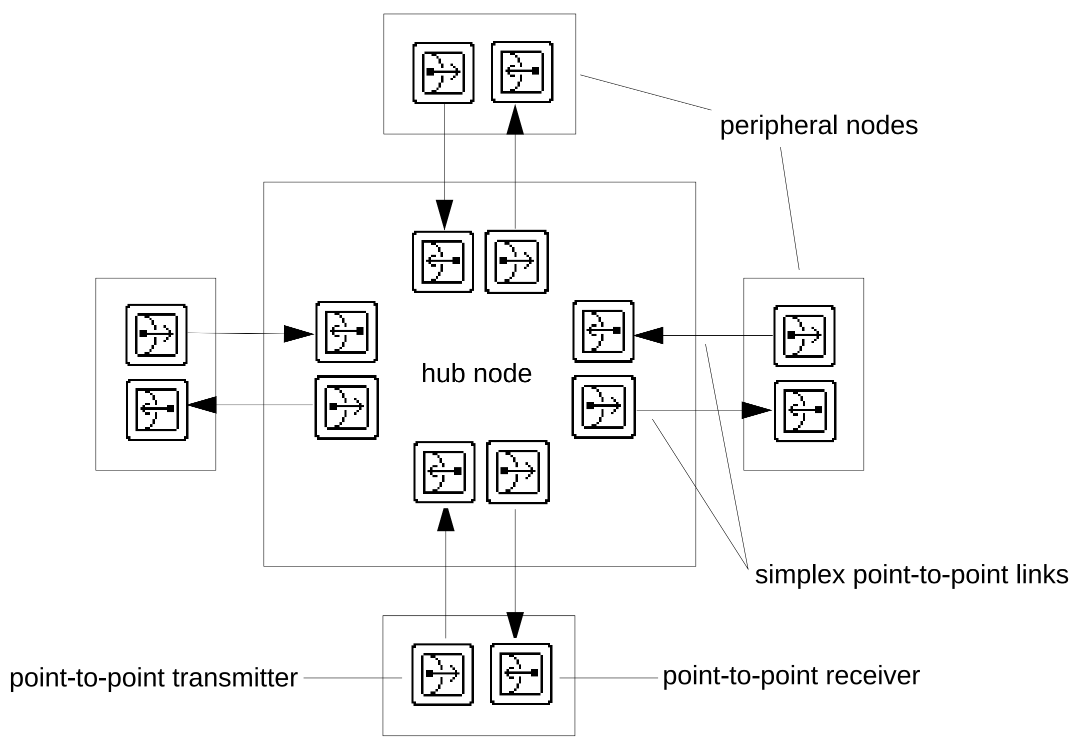


Pst.2 Designing the Model

Consider the following concepts when designing the model:

1) *Network topology and the physical communications medium*

The network is to consist of four nodes connected to a hub node by point-to-point links. An OPNET simplex point-to-point link is a one-way path originating at a point-to-point transmitter and ending at a point-to-point receiver. Two-way links consist of either two separate simplex point-to-point links going in opposite directions (as shown in the diagram below), or a single duplex point-to-point link connecting two transmitter/receiver pairs (the latter method is used in this tutorial). Looking only at the communications medium as represented in **`opnet`**, a transparent view of the network would look like this:



2) *The functions of the two node types*

The overall purpose of the model is to simulate packets traveling from one peripheral node to another peripheral node through the packet switching hub node. Begin with the hub node, and assume that packets containing destination addresses will arrive randomly on the four input links. The destination address is an integer value specifying a destination peripheral node. The hub node must contain a process model which can retrieve the incoming packets, read the destination address, and send the packets to the appropriate point-to-point transmitter.

3) *The method the process model uses to determine which point-to-point transmitter will address a particular peripheral node*

Recall that packet streams each have a unique index. The easiest, though perhaps not the most flexible method, is to set up direct association between the hub process outgoing packet stream indices and the peripheral destination address values. In a more adaptive model, the hub process model could maintain a table for translating destination address values to transmitter stream indices. For this problem, a direct correspondence between destination addresses and packet stream indices is appropriate. In summary, the hub node model will consist of a point-to-point transmitter/receiver pair for each peripheral node, and a process model used to relay packets from a receiver to the appropriate transmitter.

4) *The role of peripheral nodes*

The peripheral node model must generate packets, assign destination addresses, and process received packets. The peripheral node model will employ an OPNET ideal generator module to create packets. It will also employ a user-defined process model to assign destination addresses to packets and send them to the node's point-to-point transmitter. This process model will also retrieve packets arriving from the point-to-point receiver. Upon receiving a packet, the same process model will calculate the packet's end-to-end delay and write the value to a global statistic (a global statistic is a statistic that is accessible to multiple processes throughout the system).

Pst.3 Using the OPNET Simulation Kernel Manual

OPNET provides an extensive set of Simulation Kernel procedures (also known as KPs). These procedures allow access to simulation services and they allow access to, and manipulation of, model objects. The Kernel Procedures are divided into categories based on the type of service provided or the type of object they manipulate. The set of procedures within a category is called a *package*. The OPNET *Simulation Kernel* manual serves as a reference of all the Kernel Procedures. Each chapter in the OPNET *Simulation Kernel* manual contains a description of the Kernel Procedures within one package. The chapters are arranged alphabetically by package name. The beginning of each chapter contains an overview of the available procedures; the new user should browse through the overview of each chapter to get an idea of the services available. Some of the Kernel Procedures return or take as arguments special OPNET predefined data types. These types are described in *Chapter Intro* of the OPNET *Simulation Kernel* manual.

Kernel procedure names follow a convention which identifies the package to which they belong and the purpose they serve. All Kernel Procedures names begin with the prefix "**op_**" which quickly distinguishes them from system library and user functions. Following the "**op_**" prefix is the abbreviated package name. For instance, procedures in the **Intrpt** package will always be prefixed "**op_intrpt_**"; procedures in the **Strm** package will always be prefixed "**op_strm_**". The remainder of the procedure name identifies its purpose.

The division of Kernel Procedures into separate packages makes the process of locating a particular procedure easier. A user may have need of a particular service, but may not know the name of the Kernel Procedure which provides that service. For example, the user creates a process which at some point must create a packet. The user does not know if a Kernel Procedure exists for creating packets, but can quickly find out by searching in the package which would most likely contain such a procedure. Package names reflect the type of object or service the procedures relate to; so, the logical starting point in the search for a procedure which creates packets would be the **Pk** package.

The manual provides a two-page format for describing each Kernel Procedure. The format is divided into sections which present a variety of information about the procedure. These sections include arguments and return value specifications, a synopsis ("abstract"), detailed information about the procedure's operation and effects, a summary of the intended purpose or typical applications of the procedure, and lists of potential errors and cross-references to other KPs. Dividing the procedure documentation into distinct sections allows information to be found quickly.

The last chapter in the OPNET *Simulation Kernel* manual contains a list of predefined OPNET symbolic constants. These symbolic constants are used as arguments to Kernel Procedures, or for testing the return value of a procedure. Familiarization with the different categories of symbolic constants will help the user better understand the use of many of the Kernel Procedures.

Pst.4 Entering the Model

Creating a Packet Format

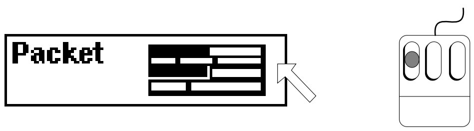
The previous tutorials have addressed problems which consider only the sizes of packets. For many modeling problems, packets must contain information in addition to just their size. For this reason, OPNET supports packets with user-specified fields which may contain virtually any information. The packet fields are identified either by an integer index or a field name. A particular combination of fields assigned to a packet constitutes the packet's format. Packet formats with named fields can be predefined in the Packet Format Editor, and referenced in a process model. Packets in this tutorial will contain destination addresses Before defining the node and process models, the Packet Format Editor will be used to create a packet format containing a destination address field.

TRY IT…*Getting started*

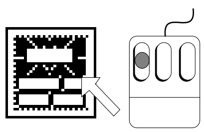
1) If **opnet** is not running, start the program.

```
% opnet
```

2) Left-click the **`Open Packet Format Editor`** tool button.



3) Create a packet field by clicking the **`Create New Field`** action button and left-clicking in the workspace.



4) Right-click on the packet field to open the packet field attributes dialog box.

The **Packet Field Attributes** dialog box contains a number of attributes, including the following:

Packet Field Attributes	
Cell Name	Description
Field Name	The name to be used when referring to this field.
Type	The data type, which may be an **integer**, **double**, a data structure pointer, a packet, or unspecified information.
Size	The simulated size of the field, measured in bits.
Default Value	The value assigned to the field when the packet is created (if **Default Set** has the value **set**).
Set at Creation	Either **set** or **unset**; if unset, the field will not be assigned the default value when the packet is created.

Process models reference the packet format in order to identify and access a packet's fields. Once written to disk, a packet format can be specified as an attribute in an ideal generator module so that packets created by the generator will be formatted accordingly. Packet formats can also be used in calls to certain Kernel Procedures which deal with packets.

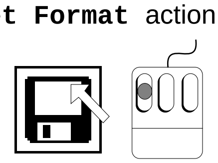
TRY IT…*Define and write the packet format, then quit*

1) Enter the following values into the dialog box.

- **a)** Change the **name** attribute to "**dest_address**".
- **b)** Change **type** to "**integer**".
- **c)** Change **size** to **0**.
- **d)** Change **Set at Creation** to "**unset**".

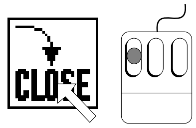
Note: Packets in this tutorial require a single field for containing the destination address. The field size is set to zero since it is not of importance to this model.

2) Activate the **Write Packet Format** action button.



3) Name the file "**pksw_format**" and press **<Return>**.

4) Activate the **`Close Packet Format Editor`** action button.

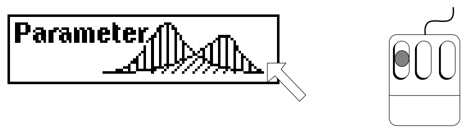


Creating a Link Model

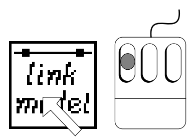
Next the Parameter Editor will be used to create a link model for connecting the nodes in the Network Editor. This model will use the packet format just created.

TRY IT…*Define the link model*

1) Open the Parameter Editor by clicking the Parameter tool button.



2) Activate the **`Edit Link Model`** action button.



➡ The link model dialog box appears.

3) In the **`Link Types`** table, change the **`Supported`** value to “**`no`**” for the **`ptsimp`**, **`bus`**, and **`bus tap`** types.

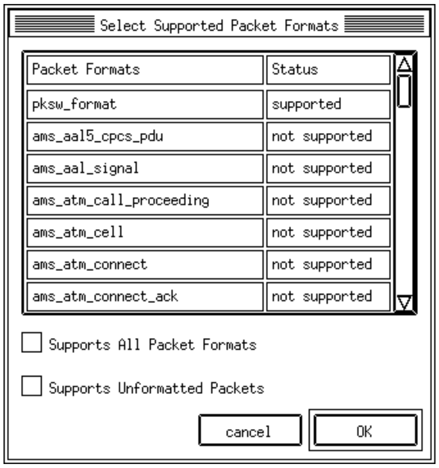
➡ This model will only support the pt-pt duplex type.

4) Change the **packet formats** attribute in the attribute table:

a) If necessary, scroll to display the **packet formats** attribute, then click in the Initial Value column to display the **Select Supported Packet Formats** dialog box.

b) Click on both check boxes to turn off **Supports All Packet Formats** and **Supports Unformatted Packets**.

c) If necessary, scroll to display the **pksw_format** format, then change the status to "**supported**".



d) Close the dialog box by clicking on the **OK** button.

5) Change other attributes in the attribute table:

a) Change the **data rate** attribute to "**9600**".

b) Change **ecc model** to "**dpt_ecc**".

c) Change **error model** to "**dpt_error**".

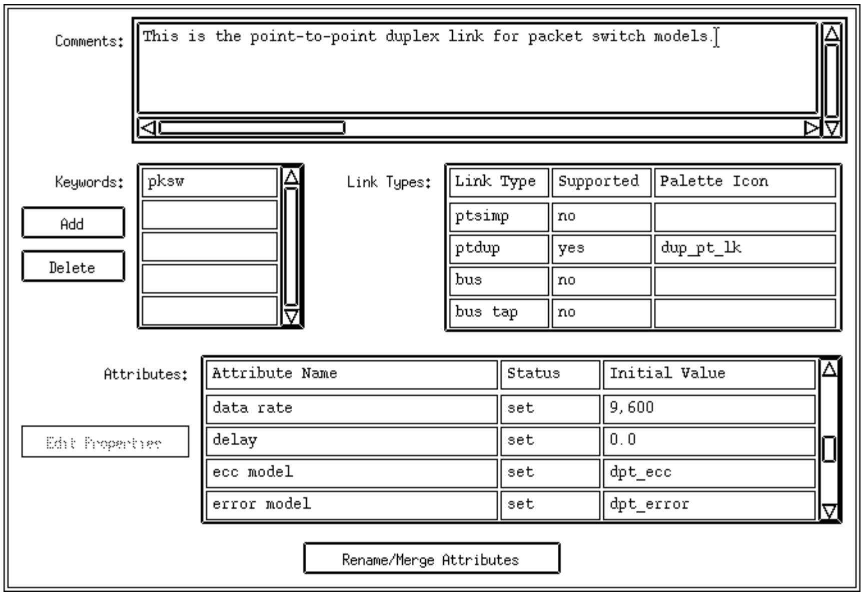
d) Change **propdel model** to "**dpt_new_propdel**".

e) Change **txdel model** to "**dpt_txdel**".

6) Add a keyword called "**pksw**".

7) Add a comment to describe the link.

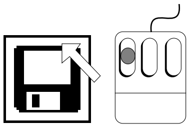
➡ The dialog box should look like this:



The link model is defined. Now write it and close the Parameter Editor.

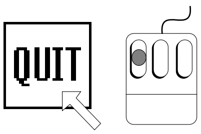
TRY IT…*Write the link model, then close*

1) Activate the **`Write Parameter Model`** action button.

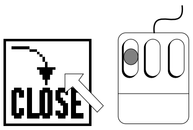


2) Name the file "**`pksw_link`**" and press **`<Return>`**.

3) Left-click on the **`Quit`** action button to exit the **`Edit Link Model`** mode.



4) Activate the **`Close Parameter Editor`** action button.

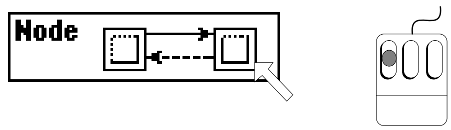


Creating the Hub Node Model

Sometimes the structure of a node model influences the design of underlying process models. For such cases, it helps to create the node model first to see how the process model(s) fit into the picture. It is important to ensure that stream and statistic indices specified with the Node Editor correspond to the index values used in underlying process models. Start by defining the hub node.

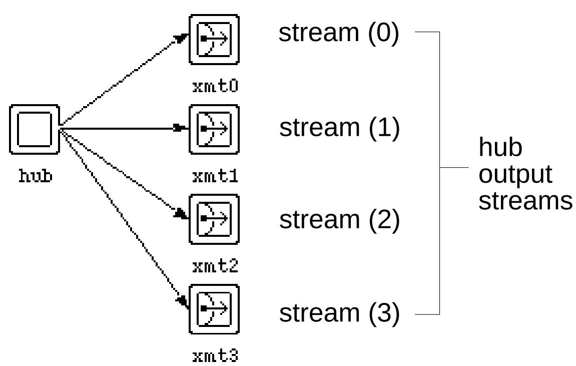
TRY IT…*Open the Node Editor and create the hub node model*

1) Open the Node Editor by clicking the Node Editor tool button.



2) Create hub and transmitter modules as shown, using the following steps:

- **a)** Use the **`Create Processor`** action button to create one processor module.
- **b)** Use the **`Create Pt-Pt Transmitter`** action button to create four point-to-point transmitter modules.
- **c)** For each module, open the attribute dialog box and change the **`name`** attribute as shown below.
- **d)** Use the **`Create Packet Stream`** action button to connect the modules as shown below. Start with the stream from **`hub`** to **`xmt0`**, working down to **`xmt3`**. **opnet** assigns the default stream indices. Going in numerical order ensures that each transmitter is automatically assigned a logical source stream index as shown above.



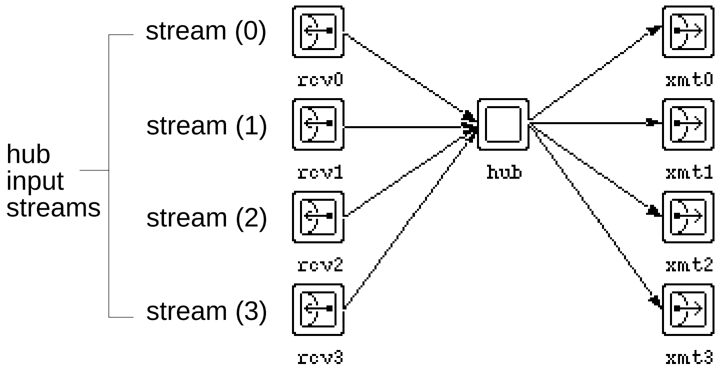
Note: you can change source stream indexes by editing the **`src stream`** attribute in the stream's attribute dialog box. Notice that index numbers become unavailable once they are assigned.

3) Create receiver modules as shown, using the following steps:

a) Use the **`Create Pt-Pt Receiver`** action button to create four point-to-point receiver modules opposite the transmitter modules.

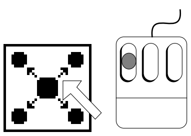
b) For each receiver module, open the attribute dialog box and change the **`name`** attribute as shown below.

c) Connect the modules with packet streams as shown. Again, going in numerical order from **`rcv0`** to **`rcv3`** ensures that each transmitter is automatically assigned a logical source stream index as shown below.



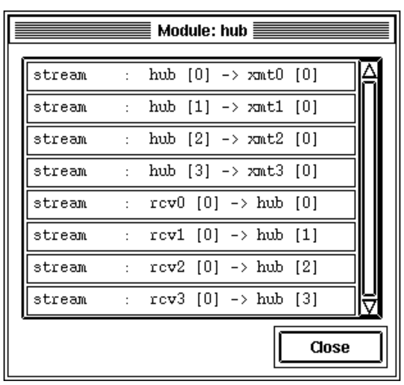
4) Check the stream assignments, using the following steps:

a) Activate the **`List Module Connectivity`** action button.



b) Move the cursor (the box outline) over the **hub** processor module and click.

➡ The list of streams connecting to the **hub** module appears.



5) If necessary, correct the streams by changing the **`src stream`** and **`dest strm`** attributes of the packet streams.

Now that you have created the hub node model, you need to set the channel data rate to **9600 bps** for each transmitter and receiver. The efficient method is to use the **`Also Apply Changes to Selected Objects`** feature to change the value of the **`data rate`** attribute for all these objects in one operation. Follow these steps.

TRY IT…*Set the channel data rates*

1) Left-click on each transmitter and receiver module icon to select it. Be sure the “selection handles” are at the corners of each icon.

2) Choose one module:

- **a)** Open the attribute dialog box by right-clicking on the icon.
- **b)** Click on the value of the **`channel`** attribute.
  - ➡ A dialog box describing the channel appears.
- **c)** Change the **`data rate`** to “**`9600`**”.
- **d)** Close the **`Channel Attributes`** dialog box by clicking on the **`OK`** button.
- **e)** Click on the **`Also Apply Changes to Selected Objects`** check box.
- **f)** Close the **`Attributes`** dialog box by clicking on the **`OK`** button.
- ➡ The channel data rate for all transmitters and receivers is **`9600`**. You can verify this by examining the attribute lists for one or two of the objects.

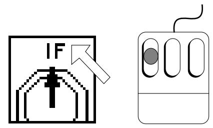
3) Left-click on a blank area of the workspace to deselect all modules.

- ➡ This is a good practice, because it prevents you from unintentionally modifying a module.

The last step in defining the node model is to specify the node types supported and a keyword.

TRY IT…*Define the node model interface attributes*

1) Activate the **`Edit/View Model Attr. Interfaces`** action button.



➡ The **`Model Attribute Interfaces`** dialog box appears.

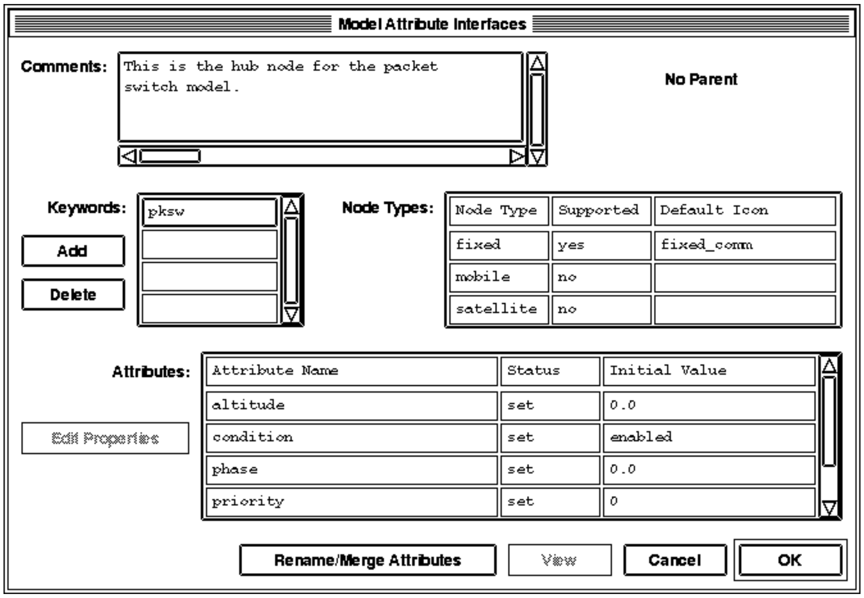
2) In the **`Node Types`** table, change the **`Supported`** value to “**`no`**” for the **`mobile`** and **`satellite`** types.

➡ This model will now only support the fixed node type.

3) Add a keyword called “**`pksw`**”.

4) Add a comment to describe the node.

➡ The dialog box should look like this:



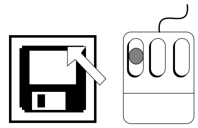
5) Save your changes by clicking on the **`OK`** button.

The hub node definition is complete except for specifying the process model for the **`hub`** processor module (we’ll do this in a later step). For now, just write the node model to disk.

Packet Switching Tutorial

TRY IT…*Write the node model*

1) Activate the **`Write Node Model`** action button.



2) Name the file “**`pksw_hub`**” and press **`<Return>`**.

You will do more work with this node model later (after developing a process model), so this time you will not close the Node Editor after writing the model.

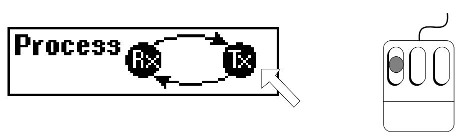
Creating the Hub Node’s Process Model

The purpose of the hub process is to relay packets to the appropriate point-to-point transmitter. Recall that the **`hub`** processor module is connected to the point-to-point transmitters and receivers by packet streams. The hub process will receive an interrupt whenever a packet arrives from a point-to-point receiver. This is the only expected type of interrupt; therefore, the hub process **`FSM`** can be defined using two states: an unforced idle state to rest between events, and a forced state containing the code for processing packets.

Open the Process Editor and create these two states named **`"idle"`** and **`"route_pk"`** as shown on the next page. Notice that you can open the Process Editor even though the Node Editor is still open. You’ll see later how to move (or *circulate*) among the two open editors.

TRY IT…*Open the Process Editor*

1) Open the Process Editor by clicking the Process Editor tool button.



TRY IT…*Create the hub process model states*

1) Activate the **`Create State`** action button and place two states in the tool window.

2) Change the **`name`** attribute of the initial state to “**`idle`**”.

Remember to use the right mouse switch to open the attribute dialog box.

3) Change the **name** attribute of the second state to "**route_pk"** and the **status** attribute to "**forced**".

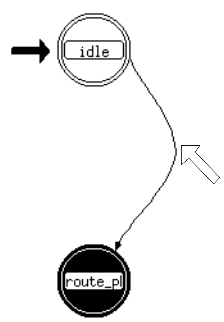
➡ The process should now look like this:

When a packet arrives from a point-to-point receiver, the process model will be invoked by a stream interrupt. A transition testing for this condition must be created.

TRY IT…*Create the PK_ARRVL transition*

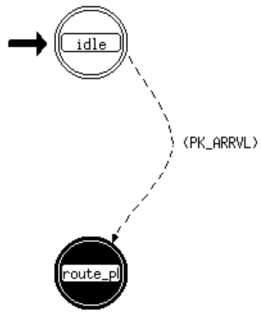
1) Activate the **Create Transition** action button and draw a transition from the **idle** state to the **route_pk** state.

Remember the technique used earlier: left-click first on the **idle** state, then at a vertex point between the **idle** and **route_pk** states (indicated by the cursor), and finally on the **route_pk** state. Right-click to end the **Create Transition** operation.

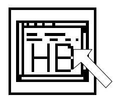


2) Change the **`condition`** attribute of the transition to "**`PK_ARRVL`**" (this macro will be defined in the next step).

Remember, you can move a condition name by left-clicking on the name and dragging it to a new location.



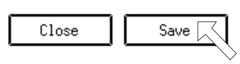
3) Activate the **`Edit Header Block`** action button.



4) Enter the following macro definition for **`PK_ARRVL`**.

```
#define PK_ARRVL (op_intrpt_type () == OPC_INTRPT_STRM)
```

5) Click on the **`Save`** button.

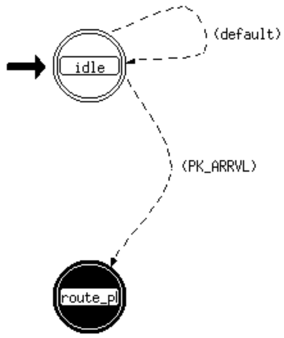


The **`PK_ARRVL`** condition compares the delivered interrupt type with the OPNET predefined symbolic constant **`OPC_INTRPT_STRM`**, which indicates a stream interrupt. This is the only expected type of interrupt in this model, but as a safeguard against run-time **missing transition errors**, unforced states sometimes have a default transition looping back on themselves.

TRY IT…*Create the unforced state default transition*

1) Activate the **`Create Transition`** action button and create a transition from the **`idle`** state and back to itself.

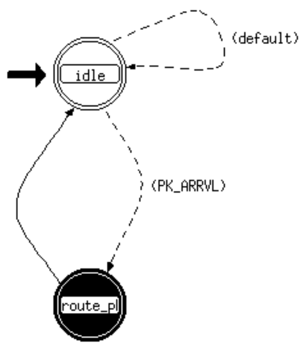
2) Change the **`condition`** attribute to "**`default`**":



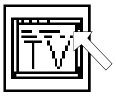
The **`FSM`** must transition back to the **`idle`** state after the executives in the **`route_pk`** state have executed.

TRY IT…*Complete the FSM diagram*

1) Create an unconditional transition from the **`route_pk`** state back to the **`idle`** state.



2) Left-click on the **Edit temporary variables** action button.

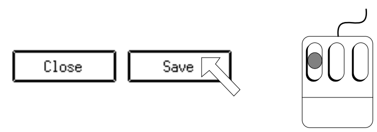


3) Enter these two declarations:

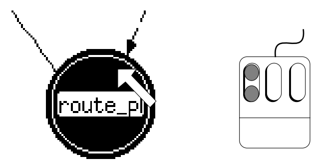
```
Packet *pkptr;
int    dest_address;
```

The first variable is for storing a pointer to packets; the second is for storing the destination address found in packets. These variables are placed in the **temporary variable block** because the values they hold do not need to be retained between invocations of the process.

4) Click on the **`Save`** button.



5) Open the **enter executives block** for **`route_pk`** by double-clicking on the top half of the state.



6) Enter the following code:

```
pkptr = op_pk_get (op_intrpt_strm ());
op_pk_nfd_get (pkptr, "dest_address", &dest_address);
op_pk_send (pkptr, dest_address);
```

7) Left-click on the **`Save`** button.

When the **FSM** enters the **`route_pk`** state, it must first retrieve the arriving packet from the appropriate input stream. The first line of the **enter executives block** accomplishes this.

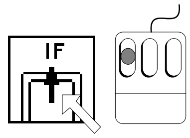
The ***op_pk_get()*** Kernel Procedure takes as an argument the index of the stream from which the packet is to be retrieved; the stream index is determined here with the ***op_intrpt_strm()*** procedure. This is the same technique as the one used to retrieve packets in *Chapter Bpt*. Next, the process must obtain the destination address contained in the packet. The destination address is held in the **named field** **`"dest_address"`**. The second line of the **enter executives** assigns the destination address to the **`dest_address`** temporary variable.

Recall that the destination address corresponds directly to the hub node's outgoing packet stream index. The final line of code sends the packet to the appropriate point-to-point transmitter based on the destination address.

Next define the model interface attributes for the process.

TRY IT…*Define the process model interface attributes*

1) Activate the **`Edit Process Model Interface`** action button.

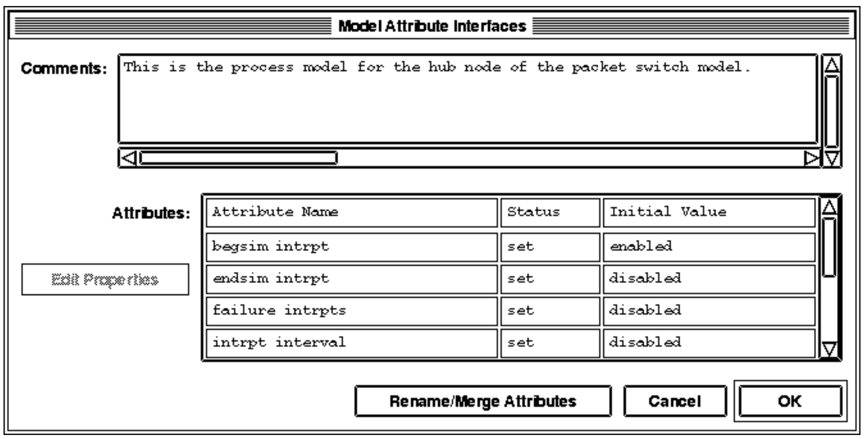


➡ The **`Model Attribute Interfaces`** dialog box appears.

2) Change the initial value of the **`begsim intrpt`** attribute to “**`enabled`**”.

3) Add a comment to describe the process.

4) Verify that the dialog box looks like this:

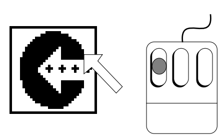


5) Save your changes by clicking on the **`OK`** button.

The hub process model definition is complete. Compile the model.

TRY IT…*Compile the process model*

1) Activate the **`Compile Process Model`** action button.



2) Supply the file name “**`pksw_hub_proc`**” and press **`<Return>`**.

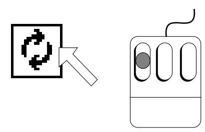
If the model does not compile, refer to the *Troubleshooting Chapter.*

Now that the hub process model is defined, you can return to the Node Editor and set the **process model** attribute of the **hub** processor module to **pksw_hub_proc**.

Follow these steps.

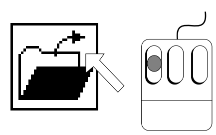
TRY IT…*Update the hub node model*

1) Move to the still-open Node Editor by left-clicking on the **Circulate** system button in the lower-right part of the window.



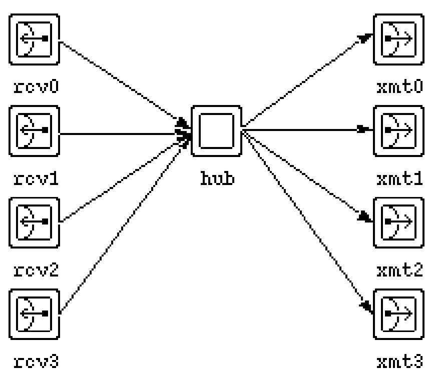
2) Read in the **pksw_hub** node model:

a) Activate the **Read Node Model** action button.



b) From the list of available files, select **pksw_hub**.

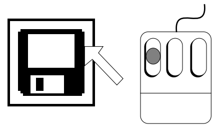
➡ The node model appears in the tool window:



3) Open the **hub** module's attribute dialog box and change the **process model** attribute to "**pksw_hub_proc**".

➡ Notice that the **begsim intrpt** attribute is now "**enabled**". This was defined in the process model attribute interface.

4) Activate the **`Write Node Model`** action button.

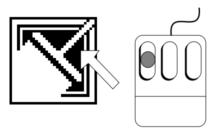


Creating the Peripheral Node Model

Leave the Node Editor open, but clear the model.

TRY IT…*Clear the node model*

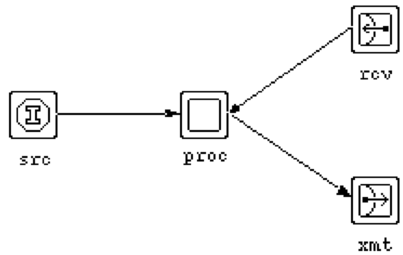
1) Activate the **`Clear Node Model`** action button.



The peripheral node will generate packets, assign destination addresses to packets, transmit the packets to the hub node, and record the end-to-end delay experienced by packets received. The peripheral node model will therefore consist of an ideal generator module, a processor module, and a point-to-point transmitter and receiver. The processor module will assign destination addresses to packets created by the generator and forward the packets to the point-to-point transmitter. When packets arrive via the point-to-point receiver, the processor module will retrieve the packets and record their calculated end-to-end delay using a global statistic.

TRY IT…*Create the peripheral node model*

1) Create the four modules as shown, using the following steps:



a) Create one ideal generator module, one processor module, one point-to-point receiver module, and one point-to-point transmitter module.

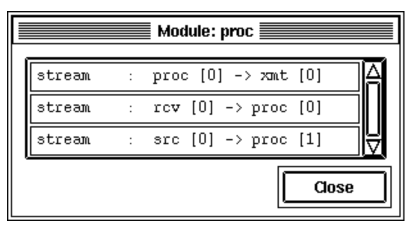
b) For each module, open the attribute dialog box and change the **`name`** attribute as shown.

c) Connect the modules with packet streams in the following order:

- rcv — proc
- src — proc
- proc — xmt.

2) Use the **`List Module Connectivity`** action button to check the assignment of the streams connecting to the **`proc`** processor module.

➡ The list of streams connecting to the **`proc`** module appears.

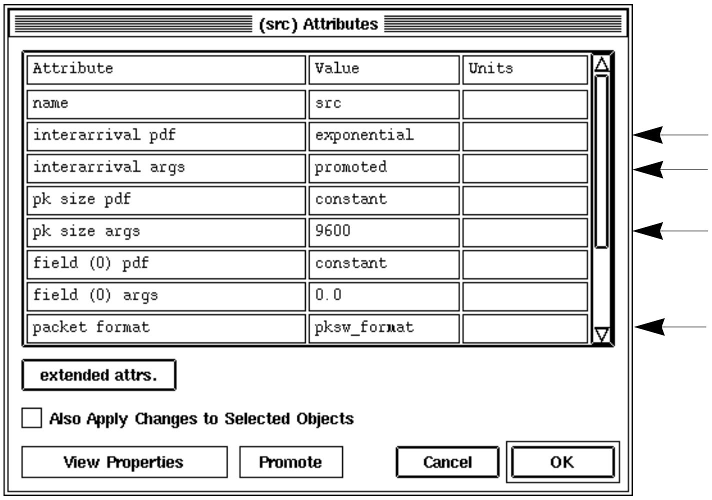

Module: proc

stream	:	proc [0] -> xmt [0]
stream	:	rcv [0] -> proc [0]
stream	:	src [0] -> proc [1]

Close

3) If necessary, correct the streams by changing the **`src stream`** and **`dest strm`** attributes of the packet streams.

4) Open the attribute dialog box for the ideal generator module and change the attributes to the values shown. To change the value for **`interarrival args`** to promoted, you can select the attribute and click on the **`Promote`** button.



5) For both the receiver and transmitter modules:

a) Click on the icon to select it. (Be sure the “selection handles” are at the corners of each icon.)

6) Open the attribute dialog box for either module and change the data rate:

a) Click on the value of the **`channel`** attribute.

➡ A dialog box describing the channel appears.

b) Change the **`data rate`** to “**`9600`**”.

c) Close the dialog box by clicking on the **`OK`** button.

d) Click on the **`Also Apply Changes to Selected Objects`** check box.

e) Close the **`Attributes`** dialog box by clicking on the **`OK`** button.

➡ The data channel rate for both modules is **`9600`**.

7) Use the **`Edit/View Model Attr. Interfaces`** action button to display the **`Model Attribute Interfaces`** dialog box and make the following changes: **a)** In the **`Node Types`** table, change the **`Supported`** value to "**`no`**" for the **`mobile`** and **`satellite`** types. ➡ This model will now only support the fixed node type. **b)** Add a keyword called "**`pksw`**". **c)** Add a comment to describe the node model.

TRY IT…*Rename attributes*

Renaming allows you to simplify complex attribute names or expand terse ones.

You might want to simplify attribute names when you know users will be working with the node model, but not with the process model — in this situation, the user does not need to see (and may be confused by) the long hierarchical attribute name. When you rename the attribute, you can hide those layers of hierarchy.

Similarly, you might want to expand a cryptic, highly-abbreviated attribute name to make it more understandable.

You can also merge multiple attributes into a single attribute, and then assign a single value. See *Chapter Fram* in the OPNET *Modeling* manual.

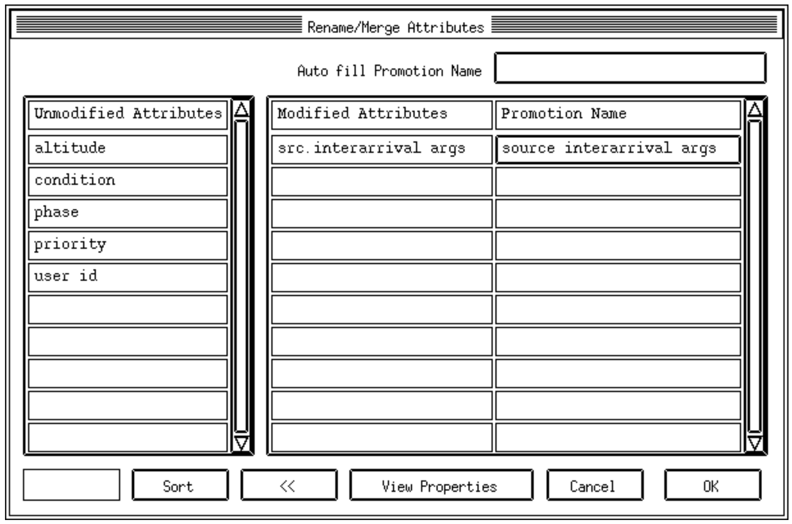
1) Verify that the **`Model Attribute Interfaces`** dialog box is displayed.
2) Click on the **`Rename/Merge Attributes`** button. ➡ The **`Rename/Merge Attributes`** dialog box pops up.

3) Rename the **`src.interarrival args`** attribute using the following steps:

a) In the Unmodified Attributes table, select the **`src.interarrival args`** attribute, then click on the **>>** button to move it to the **`Modified Attributes`** column.

b) Change the text in the **`Promotion Name`** column to "**`source interarrival args`**".

➡ The dialog box should look like this



4) Left-click **`OK`** to close the **`Rename/Merge Attributes`** dialog box.

➡ The **`Model Attribute Interfaces`** dialog box displays.

TRY IT…*Assign symbolic names to attribute values*

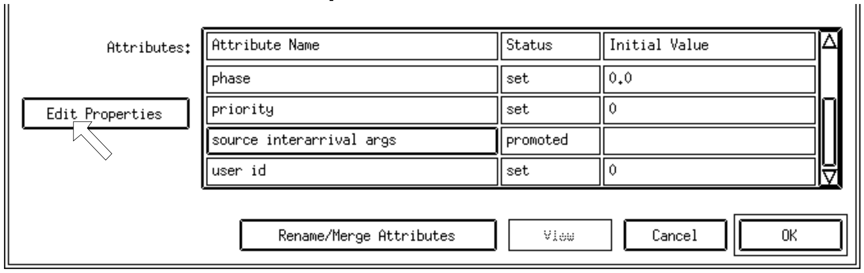
Assigning symbolic names to a set of attribute values provides a simpler, easier-to-use interface for a user, because the values you assign are displayed as options in a list. This offers three advantages:

- Options can be limited to particular, appropriate values
- Options can be assigned descriptive names, making it easier for a user to select the correct value
- Users select values from a list, avoiding typographical errors

The following exercise shows how to assign symbolic names to attribute values.

1) Make sure the **`Model Attribute Interfaces`** dialog box is still displayed.

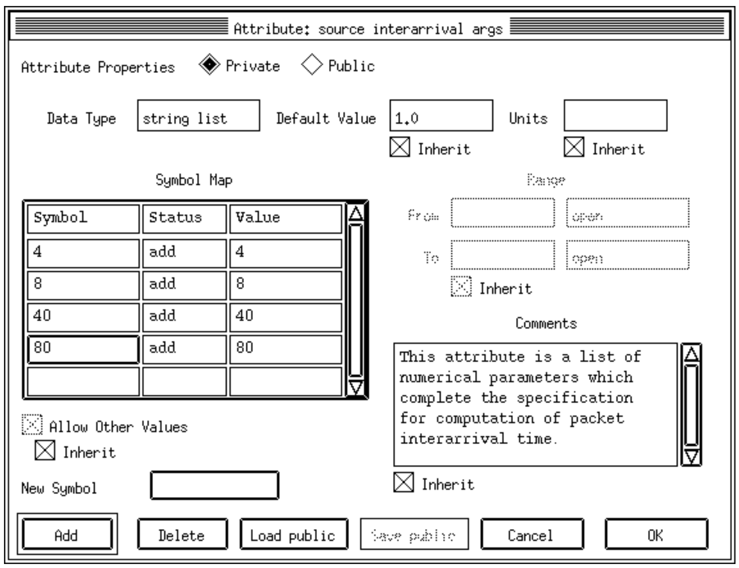
2) Select the newly-renamed **`source interarrival args`** attribute; then click on the **`Edit Properties`** button.



➡ The dialog box showing the attribute's properties pops up.

3) Create symbolic names for four values. Follow these steps.

a) Add entries in the **Symbol Map** as shown.



b) Click **`OK`** to close the dialog box.

➡ The **`Model Attributes Interfaces`** dialog box displays.

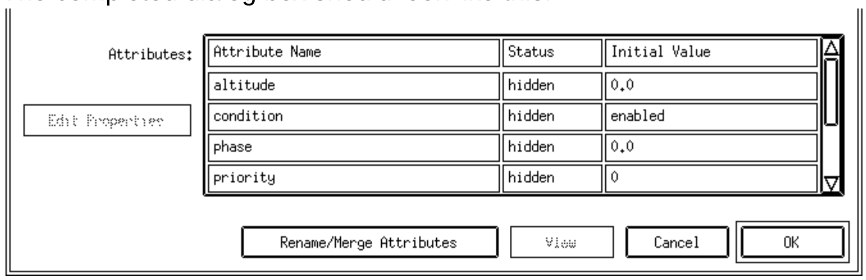
TRY IT…*Hide attributes*

1) Hide the **`altitude, condition, phase, priority,`** and **`user id`** attributes. These will cause the attributes to remain hidden on any node created with this node model. This will not affect a simulation, so they can be hidden to simplify the node interface. Follow these steps for each attribute.

a) Click and hold on the status cell in the Attributes table.

b) Select **hidden** from the pop up menu.

The completed dialog box should look like this:

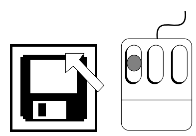


2) Click **`OK`** to close the **`Model Attributes Interfaces`** dialog box.

The peripheral node model definition is now complete except for specifying the processor module's **`process model`** attribute, which will be set after this process model has been created. Write the node model to disk.

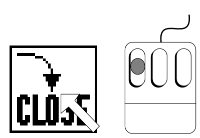
TRY IT…*Write the peripheral node model and close*

1) Activate the **`Write Node Model`** action button.



2) Name the file "**`pksw_node`**" and press **`<Return>`**.

3) Activate the **`Close Node Editor`** action button.

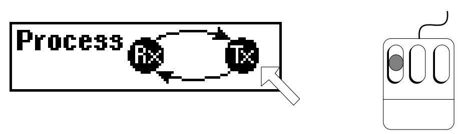


Creating the Peripheral Node's Process Model

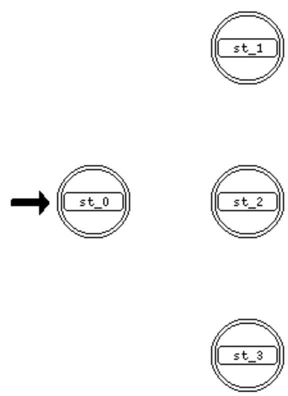
The peripheral node process model must provide two services: (1) assign destination addresses to packets to be transmitted and send the packet to the transmitter; (2) when a packet arrives from the receiver, update a global statistic used to record end-to-end delay experienced by packets. The **FSM** will have four states: an initial state, an idle state, a state for processing packets to be transmitted, and a state for processing received packets.

TRY IT…*Create the peripheral process model states*

1) Open the Process Editor by clicking the Process tool button.



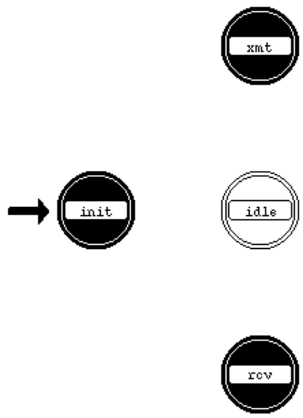
2) Activate the **`Create State`** action button and place four states in the tool window.



3) For each state, open the attribute dialog box and change the following:

- **a)** For the initial state, change the **`name`** attribute to "**`init`**" and the **`status`** to "**`forced`**".
- **b)** For the state at the top, change the **`name`** attribute to "**`xmt`**" and the **`status`** to "**`forced`**".
- **c)** For the state at the middle, change the **`name`** attribute to "**`idle`**". (Leave the **`status`** as "**`unforced`**".)
- **d)** For the state at the bottom, change the **`name`** attribute to "**`rcv`**" and the **`status`** to "**`forced`**".

The process should now look like this:



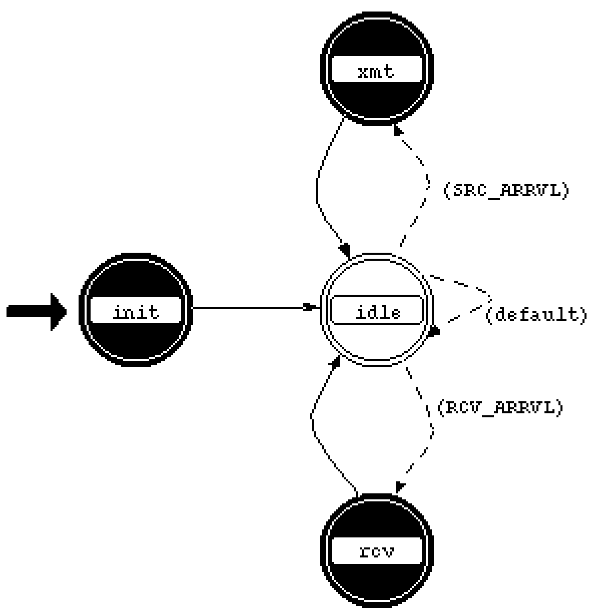
Node addresses will be integer values ranging from **0** to **3**. The process model will use OPNET's pre-defined uniform integer **PDF** for randomly assigning destination addresses. In the **init** state, the process model will load a uniform integer **PDF** using a range of **0** to **3** for possible outcomes.

In the **xmt** state, the process model will generate and assign random destination addresses to packets as they arrive from the ideal generator. The packet will then be sent out the stream connected to the point-to-point transmitter.

The **rcv** state is entered when a packet arrives on the stream connected to the point-to-point receiver. In the **rcv** state, the process model will subtract the received packet's creation time from the current simulation time and update a global statistic with the resulting end-to-end delay value.

TRY IT…*Create the peripheral process model transitions*

1) Draw the transitions between the states as shown. All transition executives are null.



2) Open the **header block** and define the transition macros for **SRC_ARRVL** and **RCV_ARRVL** as shown below.

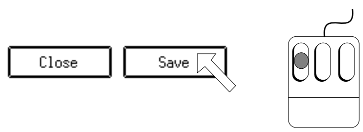
```
/* packet stream definitions */
#define RCV_IN_STRM 0
#define SRC_IN_STRM 1
#define XMT_OUT_STRM 0

/* transition macros */
#define SRC_ARRVL ( op_intrpt_type () == OPC_INTRPT_STRM && \
                    op_intrpt_strm () == SRC_IN_STRM )

#define RCV_ARRVL ( op_intrpt_type () == OPC_INTRPT_STRM && \
                    op_intrpt_strm () == RCV_IN_STRM )
```

Note the use of constant definitions of stream indices for easier code interpretation (**RCV_IN_STRM**, **SRC_IN_STRM**, and **XMT_OUT_STRM**). The stream indices correspond to the values set in the node model. Also note the backslash characters at the end of the first line of the **SRC_ARRVL** and **RCV_ARRVL** macros; these characters are needed to extend the macros to a second line of text.

3) Click on the **Save** button.



4) Using the appropriate action buttons, define the remaining variables.

The **init** state will load a uniform integer **PDF** for generating addresses from **0** to **3**. A variable must be declared that will point to the **PDF**. In addition to loading the uniform integer **PDF**, the **init** state also registers a global statistic for recording end-to-end delay. This statistic, identified by its **Stathandle**, will be written in the **rcv** state. The variable declarations for the distribution and global statistic “handles” are therefore in the **state variable block**.

a) Open the **state variable block** and declare variables for distribution and global statistic handles:

```
Distribution *\address_dist;
Stathandle    \ete_gsh;
```

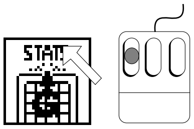
Note that the required backslash character must be placed immediately next to the variable names.

b) Open the **temporary variable block** and declare a packet pointer variable to work with code in the **xmt** and **rcv** states, then declare a temporary variable to store end-to end delay:

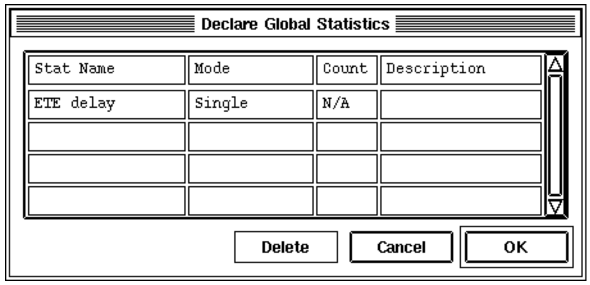
```
Packet *pkptr;
double ete_delay;
```

5) Declare the global statistic:

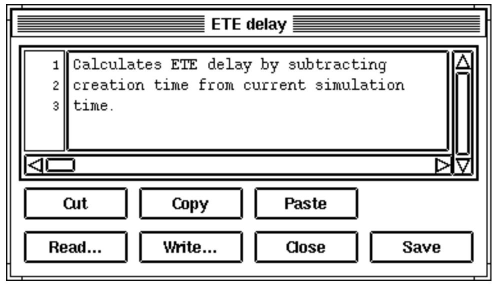
a) Activate the **Edit Global Statistics** button.



b) Enter the statistic as shown:



c) Click on the Description and enter text in the dialog box as shown:



d) Close the dialog box by clicking on the **`Save`** button.

e) Close the **`Declare Global Statistics`** dialog box by clicking on the **`OK`** button.

6) Open the **`init`** state **enter executives block** and enter these statements to load a distribution and register a global stathandle for end-to-end delay.

```
address_dist = op_dist_load ("uniform_int", 0, 3);
ete_gsh = op_stat_reg ("ETE delay", OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL );
```

7) Save the enter executives.

8) Open the **xmt** state **enter executives block** and enter the statements below.

The **xmt** state is entered when a packet arrives from the generator module. A destination address must be assigned before the packet is sent to the point-to-point transmitter. The code to do this is in the **xmt** state **enter executives**.

The first statement obtains a pointer to the packet arriving from the ideal generator. The next statement sets the value of the **dest_address** field to the value returned by the ***op_dist_outcome()*** procedure.

op_dist_outcome() returns a random number according to the distribution given as an argument. In this case, **address_dist** is a pointer to the uniform integer distribution loaded in the **init** state.

The last statement sends the packet to the output stream, which is connected to the point-to-point transmitter.

```
pkptr = op_pk_get (SRC_IN_STRM);
op_pk_nfd_set (pkptr, "dest_address", (int)op_dist_outcome (address_dist));
op_pk_send (pkptr, XMT_OUT_STRM);
```

9) Open the **rcv** state **enter executives block** and enter the statements below.

The **rcv** state is entered when a packet arrives on the stream connected to the point-to-point receiver. The **rcv** state's purpose is to calculate and record the end-to-end delay experienced by packets via a global statistic.

The first statement in the code obtains a pointer to the packet arriving from the point-to-point receiver.

The next statement calculates the end-to-end delay by subtracting the packet's creation time from the current simulation time.

The third statement writes the end-to-end delay to a global statistic, and the fourth statement destroys the packet.

When a global statistic is written, an entry is made in the **output vector file** if the global statistic is probed.

```
pkptr = op_pk_get (RCV_IN_STRM);
ete_delay = op_sim_time () - op_pk_creation_time_get (pkptr);
op_stat_write (ete_gsh, ete_delay);
op_pk_destroy (pkptr);
```

The peripheral node process model definition is complete except for defining the model interface attributes and compiling the process model.

TRY IT…*Define the interface attributes, compile, then close*

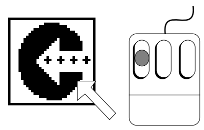
1) Use the **`Edit Process Model Interface`** action button to display the **`Model Attribute Interfaces`** dialog box and make the following changes:

a) Change the initial value of the **`begsim intrpt`** attribute to "**`enabled`**".

b) Add a comment to describe the process.

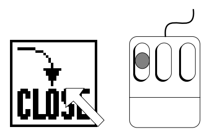
c) Close the dialog box by clicking on the **`OK`** button.

2) Activate the **`Compile Process Model action`** button.



3) Supply the file name "**`pksw_nd_proc`**" and press **`<Return>`**.

4) Activate the **`Process Editor`** action button.

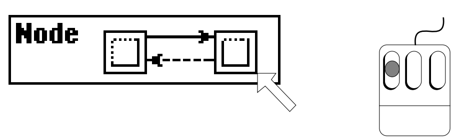


> If the model does not compile, refer to the *Troubleshooting Chapter.*

Return to the Node Editor, load the **`pksw_node`** node model, and update the process model attribute of the **`proc`** processor module.

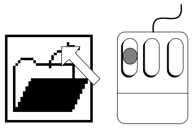
TRY IT…*Update the peripheral node model*

1) Open the Node Editor by clicking the Node Editor tool button.



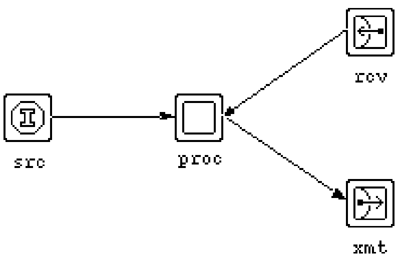
2) Read in the **pksw_node** node model:

a) Activate the **Read Node Model** action button.



b) From the list of available files, select **pksw_node**.

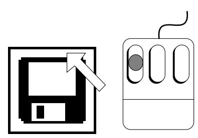
➡ The node model appears in the tool window:



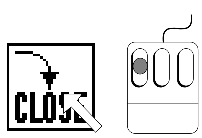
3) Open the **proc** module's attribute dialog box and change the **process model** attribute to "**pksw_nd_proc**".

➡ Notice that the **begsim intrpt** attribute is now "**enabled**". This was defined in the process model attribute interface.

4) Activate the **Write Node Model** action button, then press **<Return>** to keep the same file name.



5) Activate the **Close Node Editor** action button.

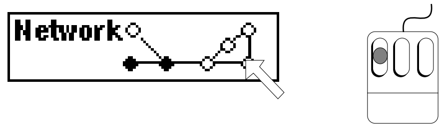


Creating the Network Model

Construct the network model at the subnet level.

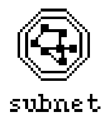
TRY IT…*Open the Network Editor and create the subnet module*

1) Open the Network Editor by clicking on the Network Editor tool button.



Remember you can optionally remove the subnet grid by activating the **`Set Grid Properties`** action button and setting the **`drawing`** property to “**`disabled`**”.

2) Using the object palette, create a subnet by clicking and dragging the following icon into the tool window.



3) Open the subnet’s attribute dialog box and change the **`name`** attribute to “**`pksw1`**”.

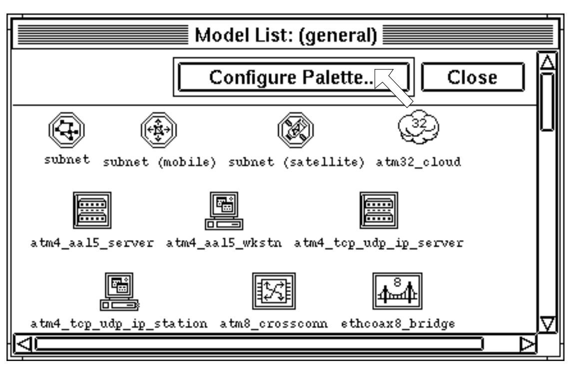
4) Double-click on the subnet to move to the subnet view.

➡ The subnet view, which will initially be blank, will appear.

Note: In this tutorial, the *extent of the subnet* (rectangular box enclosing the subnetwork) will not be discussed or used. For the remaining exercises, subnetworks may be placed anywhere within the workspace and the extent of the subnet can be ignored.

TRY IT…*Specify a keyword in the object palette*

1) In the **`Object Palette`** window, click on the **`Configure Palette...`** button.

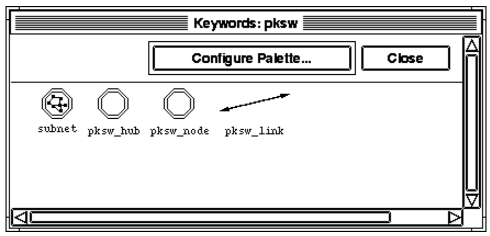


2) In the **`Configure Palette`** dialog box, make sure that the **`Keywords`** radio button is highlighted, and click on the **`Modify...`** Button. In the **`Modify Keywords`** dialog box, change the **`Status`** for **`pksw`** to “**`set`**”.

`pksw` is the keyword specified when creating the models in this tutorial.

3) Close the **`Modify Keywords`** and **`Configure Palette`** dialog boxes by clicking on their **`OK`** buttons.

➡ The object palette now displays only the subnet object and the models developed for the packet switch model.

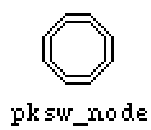


Now construct the subnet’s contents by creating fixed communication nodes and connecting them with duplex links.

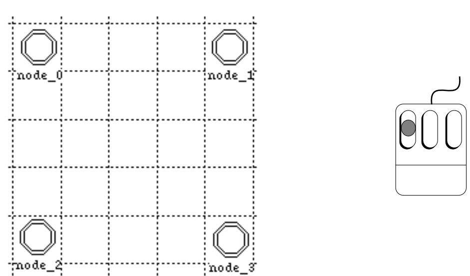
TRY IT…*Create the subnet contents*

1) Create the four peripheral nodes:

a) Find the **`pksw_node`** model in the palette.

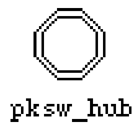


b) Click and drag the icon to the tool window and create four nodes as shown. These nodes are referred to as instances of the **`pksw_node`** model.



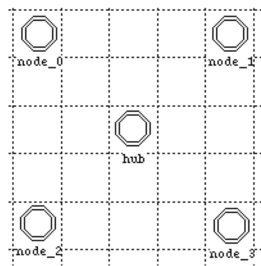
2) Create the hub node:

a) Find the **`pksw_hub`** model in the palette.



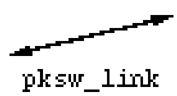
b) Click and drag the icon to the tool window to create the hub in the center of the other nodes.

c) Open the attribute dialog box for the hub and change the **`name`** attribute to “**hub**”.

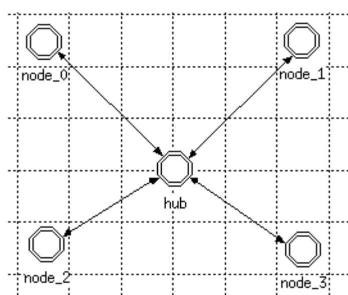


3) Connect the nodes:

a) Find the **`pksw_link`** model.



b) Draw each link from the node to the hub.

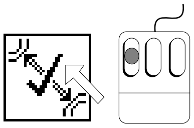


These link objects actually define separate, non-interfering paths for packets between corresponding transmitters and receivers in each connected pair of nodes.

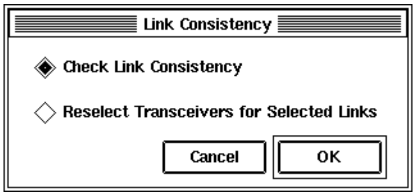
Before writing the node, check to ensure that all links are consistent.

TRY IT…*Check link consistency, write, and close*

1) Activate the **`Check Link Consistency`** action button.



2) Verify that the **`Check Link Consistency`** option is checked, then click on the **`OK`** button.

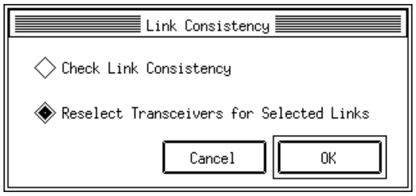


➡ A message appears in the **Message display** area.

3) If a red "**`X`**" appears over a link's icon, the link has an inconsistency. In that case, follow these steps:

a) Activate the **`Check Link Consistency`** action button again.

b) Click on the **`Reselect Transceivers for Selected Links`** check box, then click on the **`OK`** button.



➡ The red "**`X`**" disappears when the inconsistency is resolved.

> **!** If one or more red inconsistency marks are still displayed, read the following discussion and then refer to the *Troubleshooting Chapter.*

For a link between two objects to be consistent, the packet formats and data rates of the transmitter and receiver within the objects must match those defined for the link. For example, an Ethernet node containing a transmitter and receiver must connect to an Ethernet link. Although it is graphically possible to connect two Ethernet nodes via an X.25 link, it is not logically correct and would result in a simulation error. The data rates defined for Ethernet and X.25 differ and the packet formats may not be supported for the link.

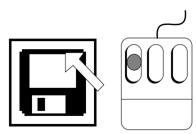
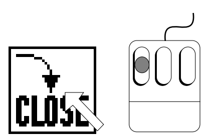
The **`Check Link Consistency`** operation lets you verify that a link is properly connected before executing a simulation. This operation checks two primary model components to ensure link consistency:

- Each node must have an available transmitter and receiver.
- The data rates of the transmitter, receiver, and link must be set to the same value.
- The packet formats defined for the transmitter, receiver, and link must have at least one format in common.

The data rate is the value specified for a transmitter, receiver, or link's **`data rate`** attribute. When different data rates are assigned to a transmitter, receiver, and link, **opnet** marks the link as inconsistent. However, you can set a link's data rate to **unspecified** to cause the simulation to automatically assign a data rate, matching the transceivers' data rates.

When there are no link inconsistencies in the packet switch model, write it to disk and close the Network Editor.

TRY IT…*Write the network model and close*

4) Activate the **`Write Network Model`** action button. 
5) Name the file "**`pksw_net`**" and press **`<Return>`**.
6) Activate the **`Close Network Editor`** action button. 

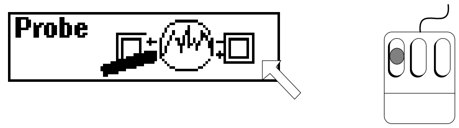
The network model definition is complete.

Specifying the Probe

To record the global statistic defined in the process model for the peripheral nodes, create a global statistic probe.

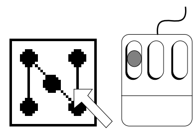
TRY IT…*Open the Probe Editor and set two probes*

1) Open the Probe Editor by clicking on the Probe button.



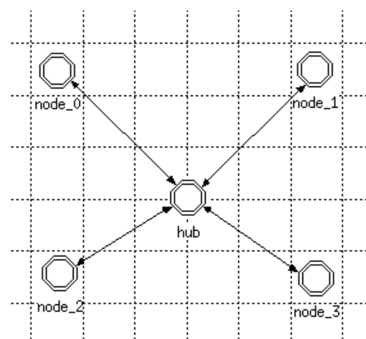
2) Set the network model:

a) Activate the **`Set Network Model`** action button.



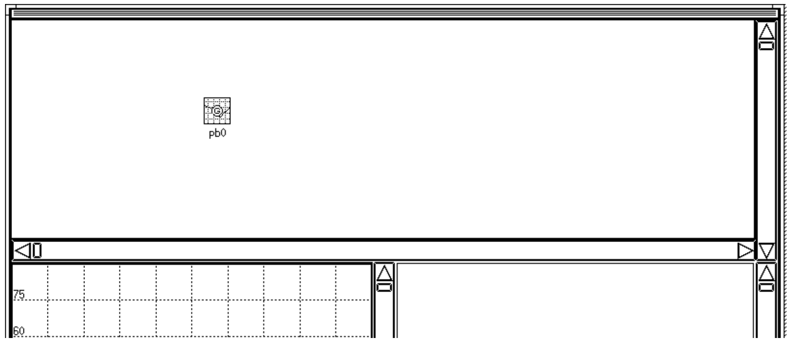
b) Select **`pksw_net`** from the menu of available files.

➡ The network model is displayed.



3) Set a global statistic probe:

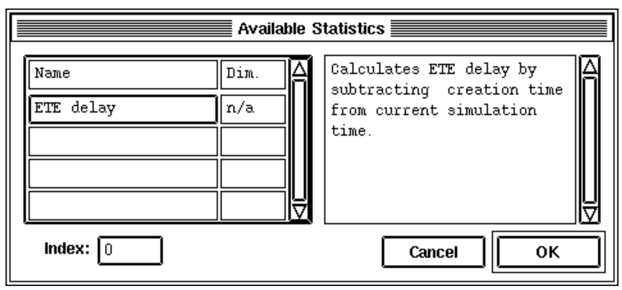
a) Using the **`Create Global Statistic Probe`** action button, create a statistic probe and place it in the Probe area of the Probe Editor window.



b) Open the probe's attribute dialog box by right-clicking on the icon.

c) Click on the **`Value`** for the **`statistic`** attribute.

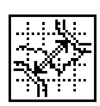
➡ A dialog box appears listing available statistics. Since there is only one statistic, this one is already selected:



d) Close the dialog box by clicking on the **`OK`** button.

e) Close the attribute dialog box by clicking on the **`OK`** button.

4) Set a link statistic probe. Using the **`Create Link Statistic Probe`** action button, create a link statistic probe and place it to the right of the global statistic probe (**`pb0`**). Then right-click on the probe to open the dialog box.



5) Change the attributes for the probe.

a) Select the probe, then click on the link between **`node_0`** and the **`hub`** to complete the **`subnet`** and **`link`** attributes.

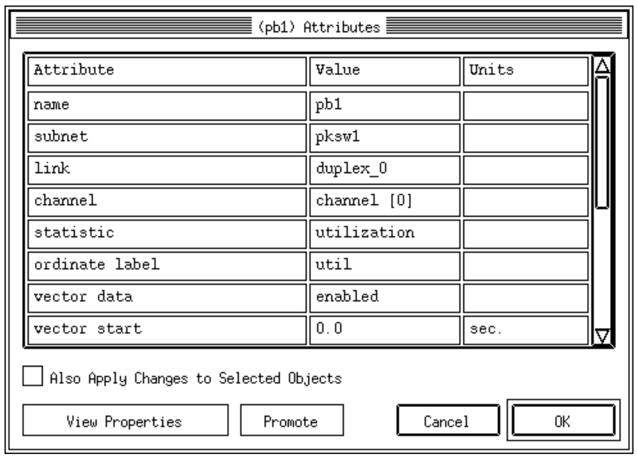
b) Left-click on the **`Value`** column of the **`channel`** row to select **`channel [0].`**

c) Left-click on the **`Value`** column of the **`statistic`** row and select **`utilization`** from the **`Available Statistics`** dialog box.

d) Enter **`util`** as the **`ordinate label`** and press< **`Return`**>.

In the Analysis Tool, the output vector for this probe will have the name given here as the **`ordinate label`** ("**`util`**").

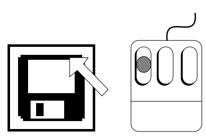
The completed dialog box looks like this:



Notice that one probe only records one statistic. Next save the probe settings to a file so that the simulation program can reference it.

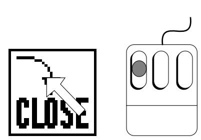
TRY IT…*Write the probe file and close*

1) Activate the **`Write Probe Model`** action button.



2) Name the file **`pksw_probes`** and press **`<Return>`**.

3) Activate the **`Close Probe Editor`** action button.



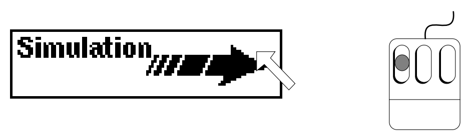
Pst.5 Executing the Simulation

The ideal generator attributes were set to create packets of a constant length. This setting in combination with the fixed data rate used by the point-to-point transmitters and receivers will result in a fixed end-to-end delay time experienced by packets. However, if two or more packets are sent to a transmitter rapidly enough, some of the packets will be delayed in the transmitter's queue. So, the one varying parameter that will affect end-to-end delays is packet interarrival time. Set up two simulations, one with slow arrivals, and the other with rapid arrivals, as shown in the following diagram.

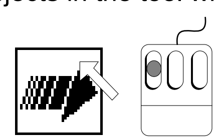
The promoted interarrival time attribute from the peripheral node model must be specified. In previous chapters, the attribute was associated with one instance of the affected model. Even though this model has four peripheral node instances, the desired attribute is still selected directly from the **`Add Attribute`** dialog box. The name of an available attribute includes a wildcard node specification so that the same value can be assigned in each of the four node instances. For example, in this network, the same interarrival attribute value needs to be assigned to all four instances of the peripheral node model.

TRY IT…*Define the simulation sequences*

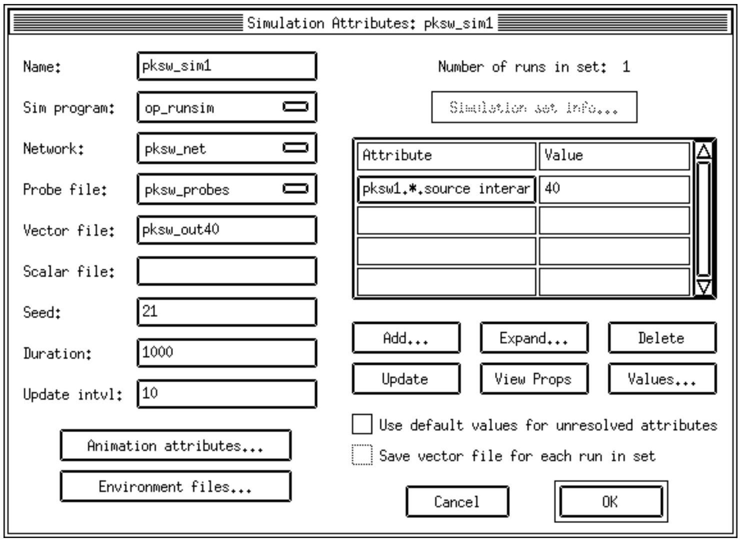
1) Open the Simulation Tool by clicking on the Simulation button.



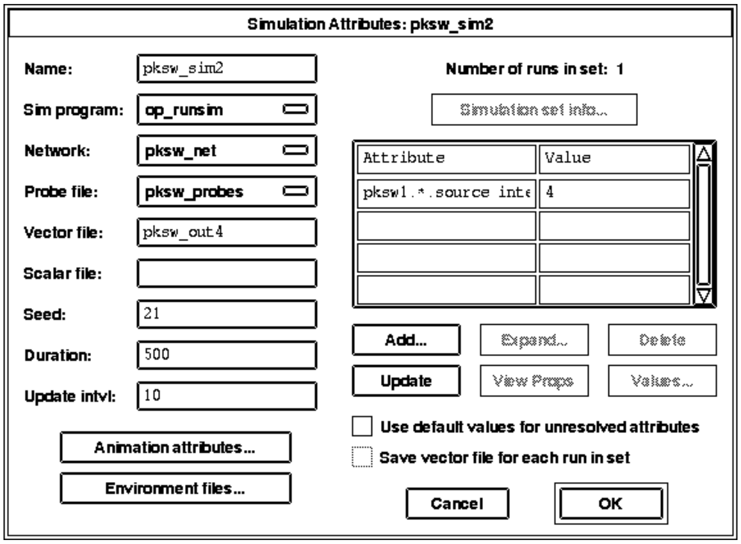
2) Activate the **`Create Simulation Set Object`** action button and place **two** simulation set objects in the tool window.



3) Open the attribute dialog box for the first object and define the simulation sequence as follows:

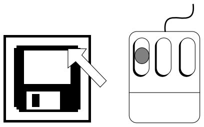


4) Open the attribute dialog box for the second object and define the simulation sequence as follows:



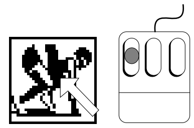
TRY IT…*Write and execute the simulation sequence, then close*

1) Activate the **`Write Simulation Sequence`** action button.



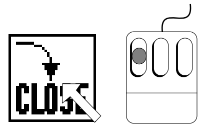
2) Name the file “**`pksw_sim`**” and press **`<Return>`**.

3) Activate the **`Execute Simulation Sequence`** action button.



➡ **opnet** displays messages to show when the simulations are complete.

4) Activate the **`Close Simulation Tool`** action button.



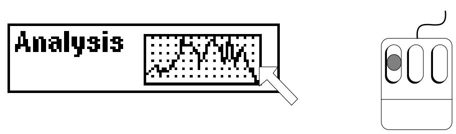
If the simulation does not execute, refer to the *Troubleshooting Chapter.*

Pst.6 Analyzing the Results

Display the recorded end-to-end delay from both simulation runs.

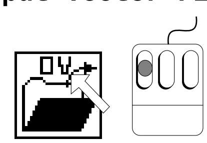
TRY IT…*Plot both vectors*

1) Open the Analysis Tool by clicking the Analysis button.



2) Open the first **output vector file**:

a) Activate the **`Open Output Vector File`** action button.

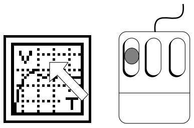


b) Select **`pksw_out40`** from the menu of statistics vectors that pops up.

This is the file with slow arrivals (large interarrival times imply slow arrivals).

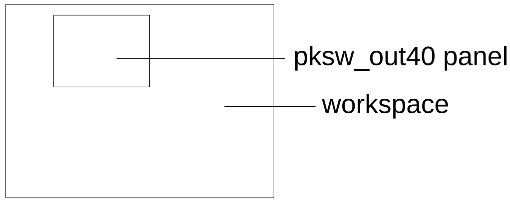
3) Draw the vector:

a) Activate the **`Create Single-vector Panel`** action button.



b) Select **`ETE delay`** from the menu of available vectors that pops up.

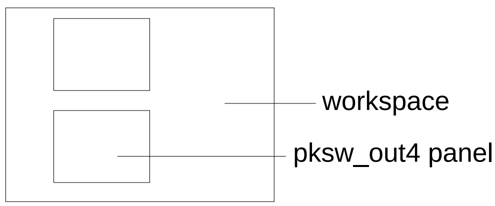
c) Place the panel within the workspace as shown:



4) Open the second output vector file by activating the **`Open Output Vector File`** action button and selecting **`pksw_out4`**.

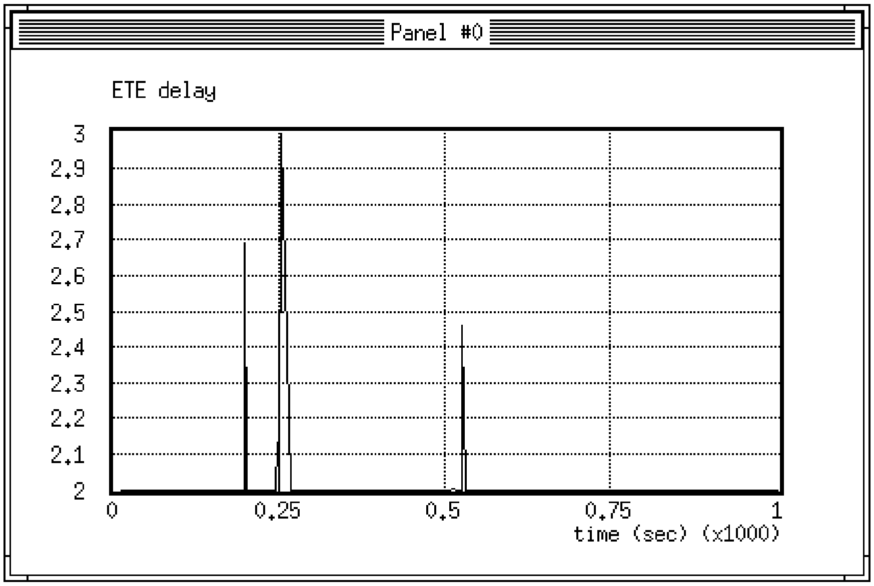
This is the file with rapid arrivals (small interarrival times imply rapid arrivals).

5) Draw the vector by activating the **`Create Single-vector Panel`** action button and selecting **`ETE delay`**. Place the panel as shown:

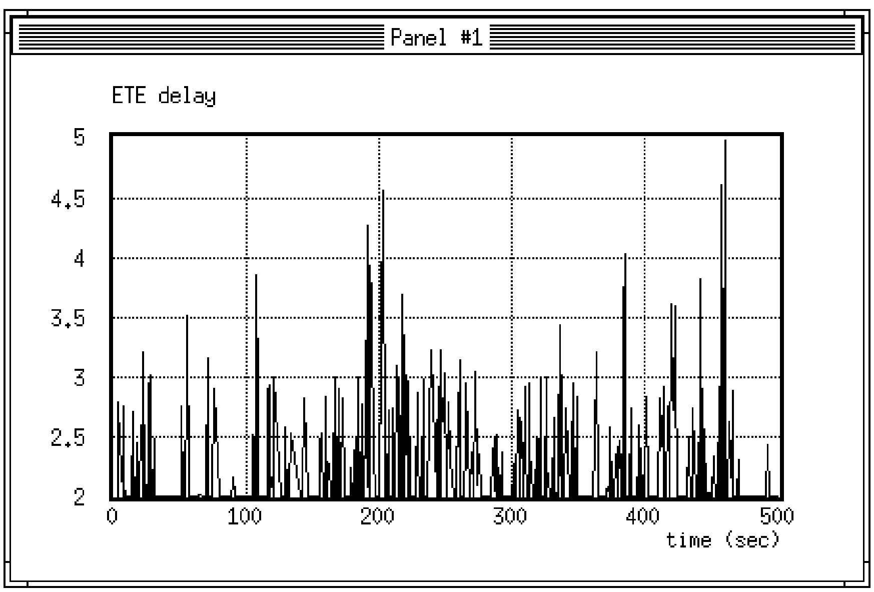


End-to-End Delays (linear graph)

pksw_out40



pksw_out4



To more clearly see the delay experienced by individual packets, change the drawing style to **`discrete`** in both panels.

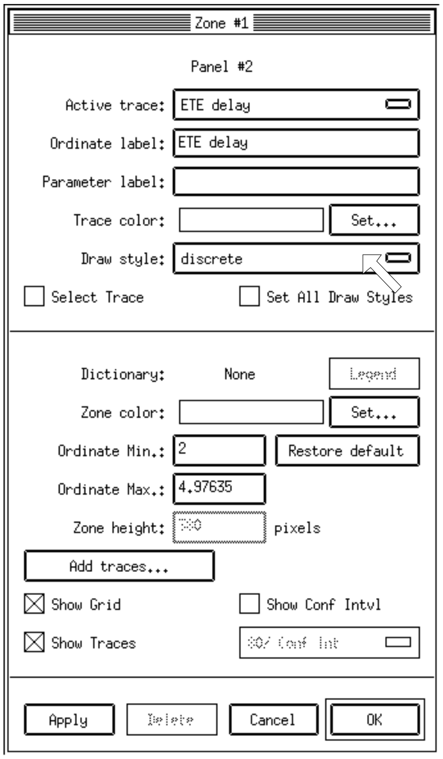
TRY IT…*Change the plotting style*

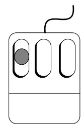
1) If the first panel is not active, left-click in it to make it active. The active panel displays a header strip.

2) Open the **`Edit Zone`** dialog box by right-clicking anywhere in the drawing area of the panel.

➡ The **`Edit Zone`** dialog box pops up.

3) Change the **`Draw style`** to "**`discrete`**".





4) Close the **`Edit Zone`** dialog box by clicking on the **`OK`** button.

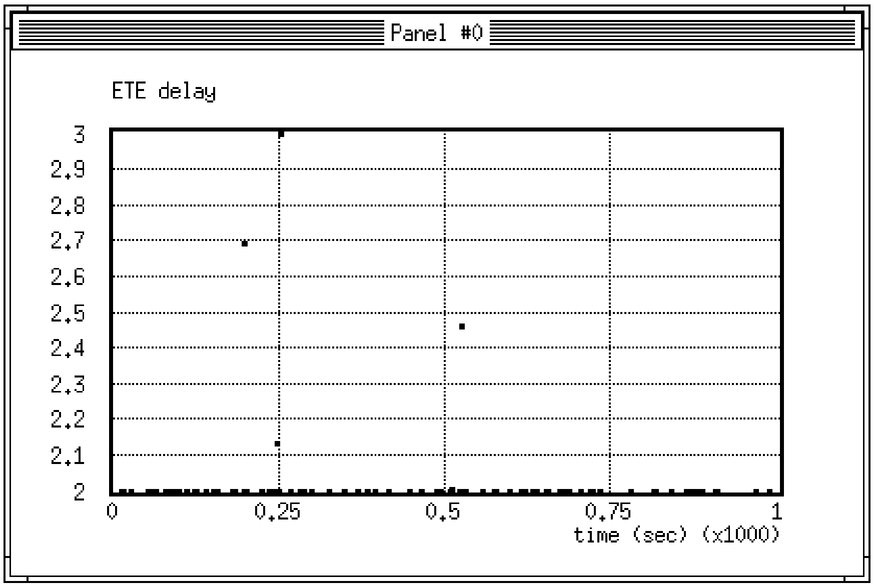
➡ The graph panel now displays a discrete graph.

5) Follow Steps 1-4 for the other panel.

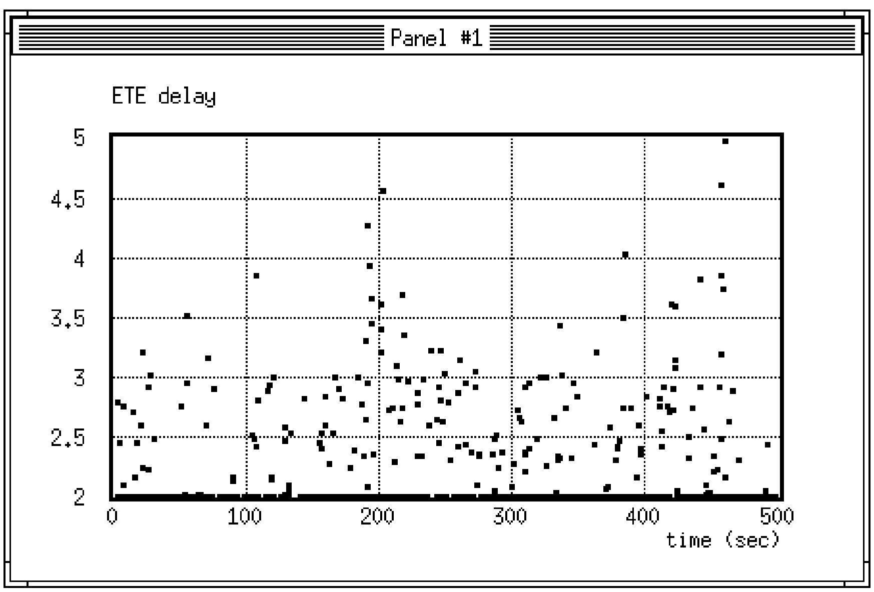
Notice the heavy concentration of two-second end-to-end delay values. Each of these points results from a packet that experiences two one-second transmission delays, but no queuing delays. This is the predominant situation in both runs, but the run with more frequent packet arrivals results in more frequent queuing delays.

End-to-End Delays (discrete graph)

pksw_out40

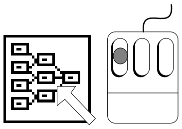


pksw_out4

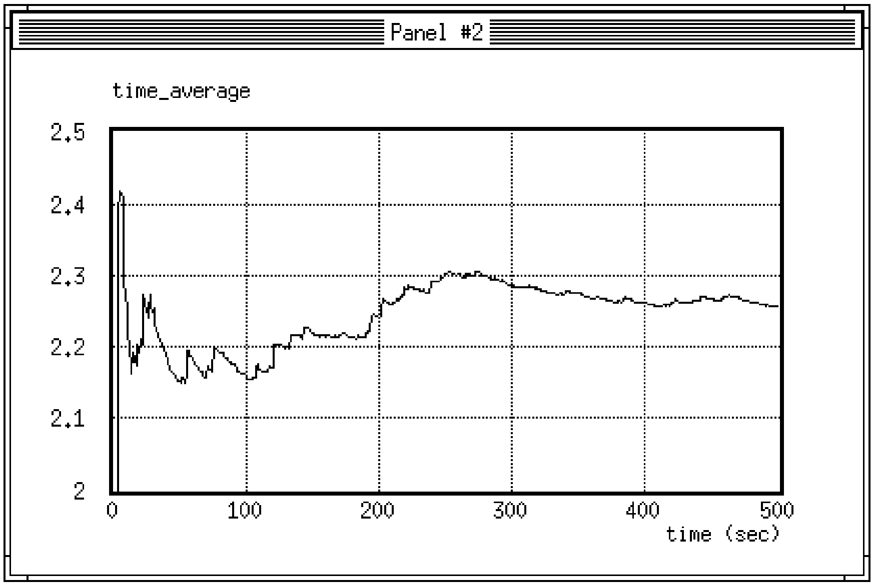


The points in the previous graphs represent individual packet delays. The Analysis Tool can be used to further examine this data using filters. The following procedure illustrates how to plot the average end-to-end delay over time.

TRY IT…*Filter a vector*

1)	Activate the **`Execute Filter Model`** action button. 
2)	Select **`time_average`** from the menu of available filters that pops up (you might need to scroll).
3)	At the prompt for a filter product label, press **`<Return>`**; the name of the filter will be used.
4)	Select **`ETE delay`** from the menu of available vectors that pops up. This vector will serve as the input of the filter.
5)	Draw the panel anywhere in the workspace.

Time Average Graph of ETE delay



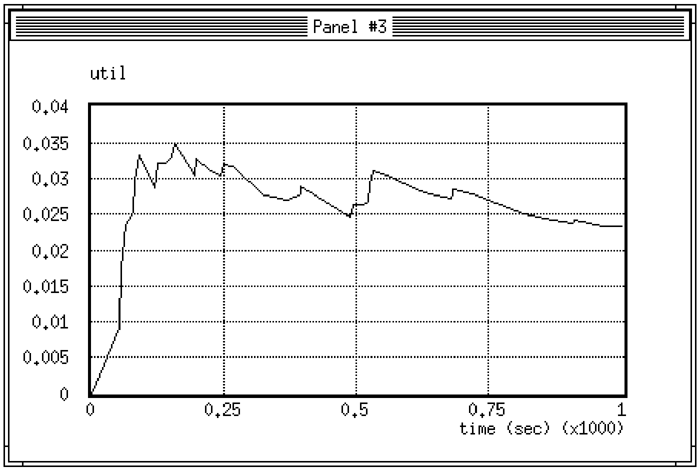
Finally, compare the utilization data gathered with the link statistic probe.

TRY IT…*Plot the utilization vectors*

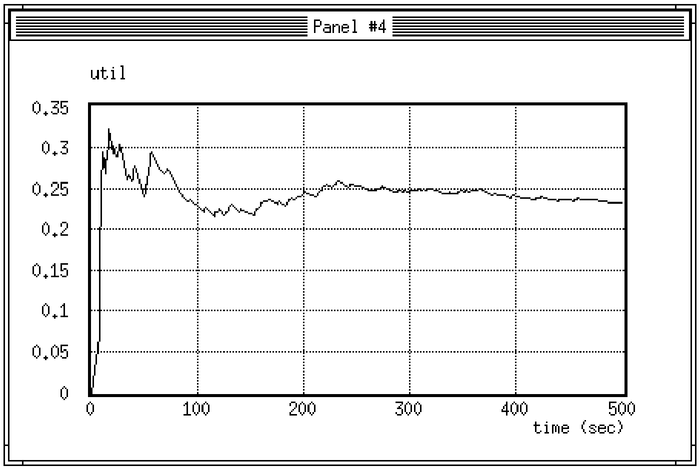
1)	Activate the **`Open Output Vector File`** action button and select **`pksw_out40`** from the menu of statistics vectors.
2)	Draw the vector by activating the **`Create Single-vector Panel`** action button and selecting **`util`**.
3)	Repeat steps 1 and 2 for the output vector file **`pksw_out4`**.

Utilization

pksw_out40



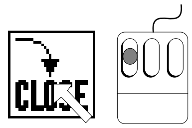
pksw_out4



Experiment with other filters if desired, then close the Analysis Tool.

TRY IT…*Close the Analysis Tool*

1) Activate the **`Close Analysis Tool`** action button.

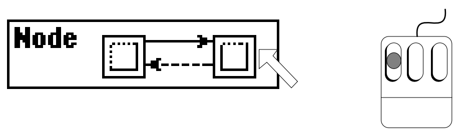


Pst.7 Enhancing the Model with an Additional Subnet

The next experiment is to enhance the model with the addition of a second star network. The two subnets will be connected via each subnet's hub node with point-to-point links. Several modifications must be made to allow addressing of packets in either subnet. The hub node model must have an additional point-to-point transmitter and receiver added to accommodate inter-net communications.

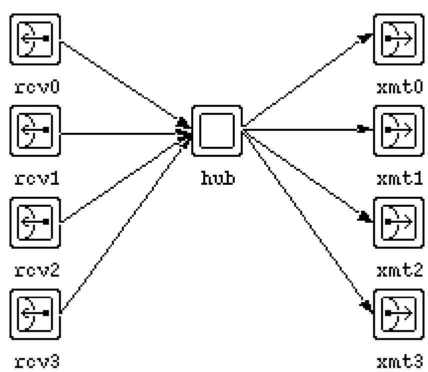
TRY IT…*Enhance the pksw_hub node*

1) Open the Node Editor by clicking the Node tool button.

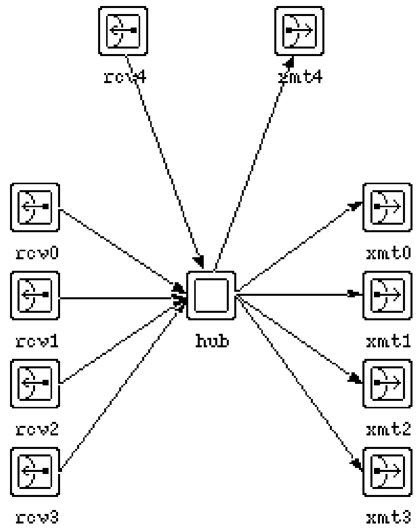


2) Read in the **`pksw_hub`** node model by activating the **`read node model`** action button and selecting **`pksw_hub`** from the list of available files.

➡ The node model appears in the tool window:



3) Add a point-to-point transmitter and receiver as shown, using the following steps:



a) Using the appropriate action buttons, create one pt-pt receiver and one pt-pt transmitter.

b) Change the **name** attributes to **`rcv4`** and **`xmt4`**.

c) Connect the new modules to the **hub** with packet streams.

➡ The new input and output streams of the **hub** module default to stream **`4`**.

4) Set the channel data rate for both new modules to 9600 bps:

a) Click on each new module to select it. (Be sure the “selection handles” are at the corners of each icon.)

b) Open the attribute dialog box of either new module by right-clicking on the icon.

c) Click on the value of the **`channel`** attribute.

➡ A dialog box describing the channel appears.

d) Change the **`data rate`** to “**`9600`**”.

e) Close the dialog box by clicking on the **`OK`** button.

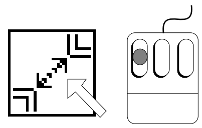
f) Click on the **`Also Apply Changes to Selected Objects`** check box.

g) Close the **`Attributes`** dialog box by clicking on the **`OK`** button.

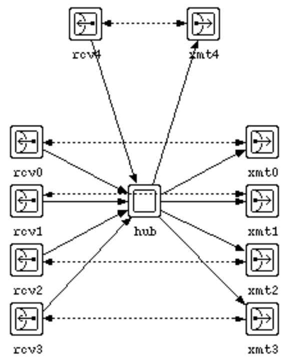
➡ The data channel rate for both the new transmitter and the new receiver is **`9600`**.

5) Logically associate each receiver with a transmitter, using the following steps:

a) Activate the **`Create Logical Tx/Rx Association`** action button.

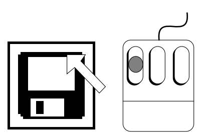


b) Connect the modules as shown:

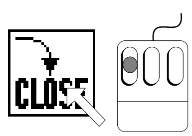


➡ This ensures that associated transmitters and receivers are used as a pair when connecting links to nodes at the network level.

6) Activate the **`Write Node Model`** action button. and press **`<Return>`** to keep the same file name. Confirm that you wish to overwrite the existing file.



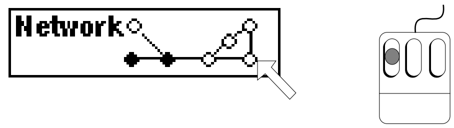
7) Activate the **`Close Node Editor`** action button.



Since an extra transmitter/receiver pair is available, the network model can be expanded to include two subnets which are connected by point-to-point links.

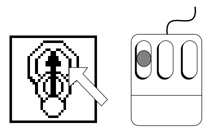
TRY IT…*Open the Network Editor and create the subnet module*

1) Open the Network Editor by clicking on the Network Editor tool button.



2) Read in the **`pksw_net`** network model by activating the **`Read Node Model`** action button and selecting **`pksw_net`** from the list of available files.

3) Activate the **`Return To Parent Subnetwork`** action button.

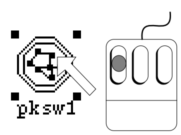


Rather than building the second subnetwork from scratch, simply copy the existing subnet.

TRY IT…*Create a new subnet by copying*

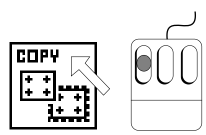
1) Copy the **`pksw1`** subnet:

a) Select the **`pksw1`** subnet by clicking on the subnet icon.



➡ Note that selection markers appear around the icon.

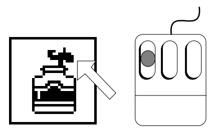
b) Activate the **`Copy`** action button.



➡ **opnet** copies the subnet to the clipboard.

2) Paste the subnet:

a) Activate the **`Paste`** action button.



➡ An outline box appears in the tool window.

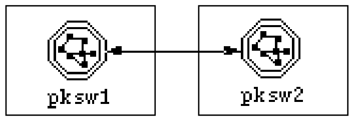
b) Place the second subnet by moving the outline box to the right of the first subnet and clicking.

➡ A dialog box appears prompting for a new name suffix.

c) Press **`<Return>`** to temporarily name the new subnet.

3) Open the attribute dialog box for the new subnet and change the **`name`** attribute to “**`pksw2`**”.

4) Using the object palette, connect the two subnets with the **`pksw_link`** duplex point-to-point link.



This connection operation searches subnet nodes for available receivers and transmitters. Only one receiver and transmitter will be available (**`xmt4`** and **`rcv4`**) in each **`hub`** node. These are the recent additions to the **`pksw_hub`** node model.

Add "**user id**" attributes as shown below so that the model can uniquely identify each subnet for addressing purposes.

TRY IT…*Set the subnet user ID's, write the model and close*

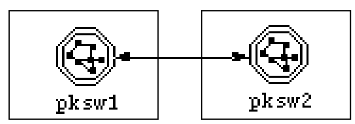
1) Change the **user id** attribute for the **pksw1** subnet:

- **a)** Open the attribute dialog box for the **pksw1** subnet.
- **b)** Click on the **Show all attributes** check box to display more attributes.
- **c)** Change the **user id** attribute to "**1**".
- **d)** Click on the **OK** button.

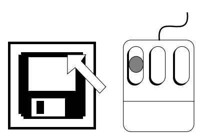
2) Change the **user id** attribute for the **pksw2** subnet to "**2**".

➡ User ID's for both subnets are assigned as follows:

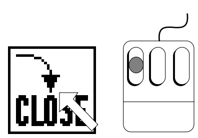
user id = 1 user id = 2



3) Activate the **Write Network Model** action button and press **<Return>** to keep the same file name. Confirm that you wish to overwrite the existing file.



4) Activate the **Close Network Editor** action button.

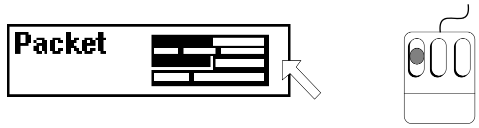


Before the new configuration can be used, the process models must be modified to handle routing of packets between the two subnets.

There are a number of ways to deal with the problem of node addressing in different subnets. Any method used must somehow identify the subnet in which a node resides. Adding another packet field which identifies the destination node's subnet is an easy way to do this.

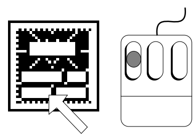
TRY IT…*Add another packet field*

1) Open the Packet Format Editor by clicking the Packet Format tool button.



2) Read in the **`pksw_format`** packet format by activating the **`Read Packet Format Model`** action button and selecting **`pksw_format`** from the list of available files.

3) Activate the **`Create New Field`** action button.

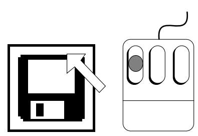


4) Set the following attribute values in the second field:

Name:	**`dest_subnet`**
Type:	**`integer`**
Size (bits):	**`0`**
Default Value:	<leave blank>
Set at Creation:	**`unset`**

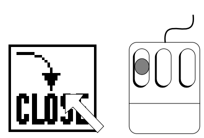
TRY IT…*Write the model and close*

1) Activate the **`Write Parameter Model`** action button.



2) Press **`<Return>`** to keep the same name.

3) Activate the **`Close Packet Format Editor`** action button.

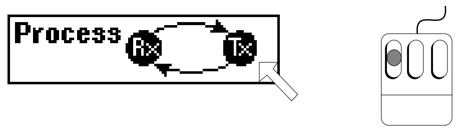


Now the process models must be modified to deal with the new configuration and packet format.

Peripheral nodes must now assign a destination subnet as well as a destination node address. The subnet address will be one of two integer values: **`1`** or **`2`** (these values correspond to a subnet's **`user id`** attribute). A new line of code in the **enter executives** for the **`xmt`** state will assign a destination subnet that randomly varies between these values for successive packets.

TRY IT…*Modify the peripheral node process model*

1) Open the Process Editor by clicking the Process tool button.



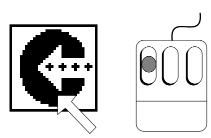
2) Read in the **`pksw_nd_proc`** model by activating the **`Read Process Model`** action button and selecting **`pksw_nd_proc`** from the available files.

3) Open the **`xmt`** state **enter executives block** and add the following line of code (indicated in bold):

```
pkptr = op_pk_get (SRC_IN_STRM);
op_pk_nfd_set (pkptr, "dest_subnet", (int)op_dist_uniform (2.0) + 1);
op_pk_nfd_set (pkptr, "dest_address", (int)op_dist_outcome (address_dist));
op_pk_send (pkptr, XMT_OUT_STRM);
```

4) Save the enter executives block text.

5) Compile the process by activating the **`Compile Process Model`** action button. and pressing **`<Return>`** to keep the same name.



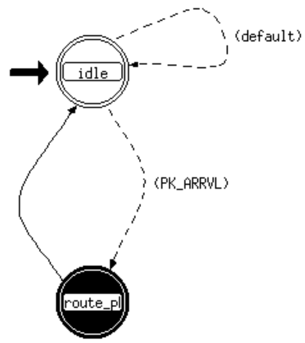
> ! If the model does not compile, refer to the *Troubleshooting Chapter.*

The hub node's process model must determine which subnet it belongs to and route packets accordingly. An initial state must be added in which the process model determines its surrounding subnet.

TRY IT…*Modify the hub node process model*

1) Read in the **`pksw_hub_proc`** model.

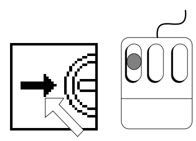
➡ The hub process appears in the tool window:



2) Create another state, using the following steps:

a) Using the **`Create State`** action button, create a new state to the left of the **`idle`** state.

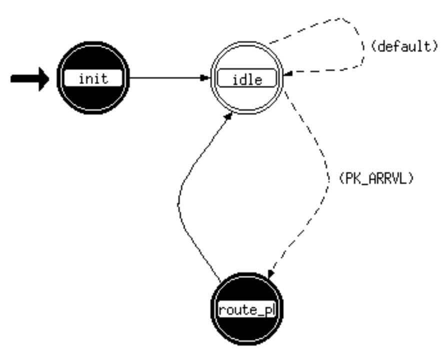
b) Make the new state the initial state by activating the **`Set Initial State`** action button and clicking on the new state.



c) Change the **`name`** attribute for the new state to “**`init`**” and change the **`state`** attribute to “**`forced`**”.

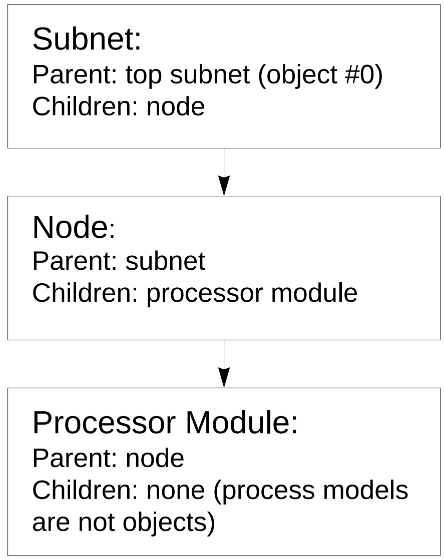
d) Create an unconditional transition from the **`init`** state to the **`idle`** state. This is the default transition.

The modified process model looks like this:



Each component of an OPNET model is an *object*. Objects are related hierarchically in a chain of parents and children; an example hierarchy is shown below.

Example Object Hierarchy



For instance, in this model the **`hub`** processor module is an object whose parent is the **`pksw_hub`** node. OPNET provides a set of Kernel Procedures for object identification and manipulation. The **`init`** state will use some of these procedures to determine which subnet the process resides in. Add the necessary variable declarations and code to the process model as shown (indicated in bold).

TRY IT…*Modify the code in the hub process model*

1) Open the **header block** and add the following code (indicated in bold):

```
#define PK_ARRVL (op_intrpt_type () == OPC_INTRPT_STRM)
#define OTHER_SUBNET_STRM 4
```

2) Open the **state variable block** and add the following code:

```
int \subnet_id;
```

3) Open the **temporary variable block** and add the following code (indicated in bold):

```
Packet *pkptr;
int    dest_address;
int    dest_subnet;
Objid  parent_subnet;
```

4) Open the **init** state **enter executives block** and add the following code (indicated in bold):

```
parent_subnet = op_topo_parent (op_topo_parent (op_id_self ()));
op_ima_obj_attr_get (parent_subnet, "user id", &subnet_id);
```

5) Open the **route_pk** state **enter executives block** and add the following code (indicated in bold):

```
pkptr = op_pk_get (op_intrpt_strm ());
op_pk_nfd_get (pkptr, "dest_subnet", &dest_subnet);
if (dest_subnet == subnet_id)
    {
    op_pk_nfd_get (pkptr, "dest_address", &dest_address);
    op_pk_send (pkptr, dest_address);
    }
else
    op_pk_send (pkptr, OTHER_SUBNET_STRM);
```

The first statement in the **init** state **enter executives** determines the OPNET object identifier of the parent subnet using chained calls to the ***op_topo_parent()*** procedure. The next statement uses this value as an argument to the ***op_ima_obj_attr_get()*** procedure to obtain the **user id** attribute of the subnet. Recall that the **user id**'s for the two subnets were set to **1** and **2** with the Network Editor.

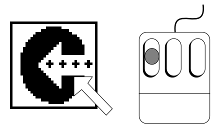
The **route_pk** state **enter executives** is modified to determine whether a packet is addressed to a node in the local subnet, or if the packet must be relayed to the other subnet. Another temporary variable is also added to store the destination subnet value contained in the packet. The **OTHER_SUBNET_STRM** symbolic constant refers to the stream attached to the newly created transmitter linked to the other subnet. The constant's definition is added at the end of the **header block.**

The modified **route_pk** state sends the packet to a local node if the destination subnet value matches the **subnet_id** obtained in the **init** state; otherwise, the packet is sent to the other subnet.

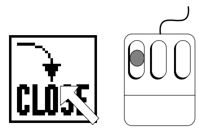
The modifications to **pksw_hub_proc** are complete. The next step is to compile the model and close the Process Editor.

TRY IT…*Compile the hub process model and close*

1) Compile the process by activating the **`Compile Process Model`** action button and pressing **`<Return>`** to keep the same name.



2) Activate the **`Close Process Editor`** action button.

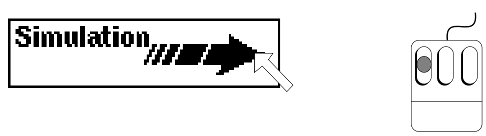


If the model does not compile, refer to the *Troubleshooting Chapter.*

All enhancements are complete. Next, enter the Simulation Tool and read in the **`pksw_sim`** sequence. Since there are now twice as many nodes, double the interarrival times and name the output files accordingly.

TRY IT…*Define the simulation sequences*

1) Open the Simulation Tool by clicking on the Simulation button.



2) Open the simulation file by activating the **`Read Simulation Sequence`** action button and selecting **`pksw_sim`**.

3) Open the attribute dialog box for **`pksw_sim1`** and make the following changes:

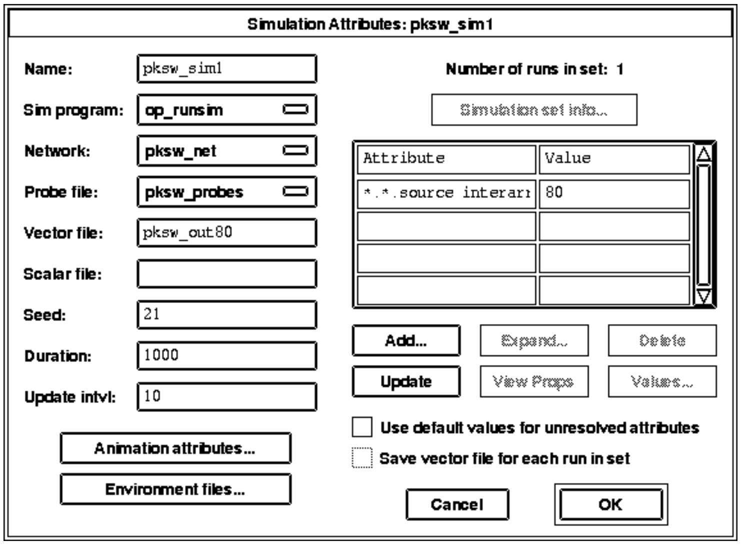
a) Change the **`Vector file`** to "**`pksw_out80`**":

b) Change the **`Attribute`** to "**`*.*.source.interarrival args`**".

Since there are now two subnets, a wildcard is used for both subnet and node levels. To make this change, delete the existing attribute, then click on the **`Add...`** button.

c) Change the **`Value`** to "**`80`**".

➡ The modified simulation attribute dialog box will look like this:



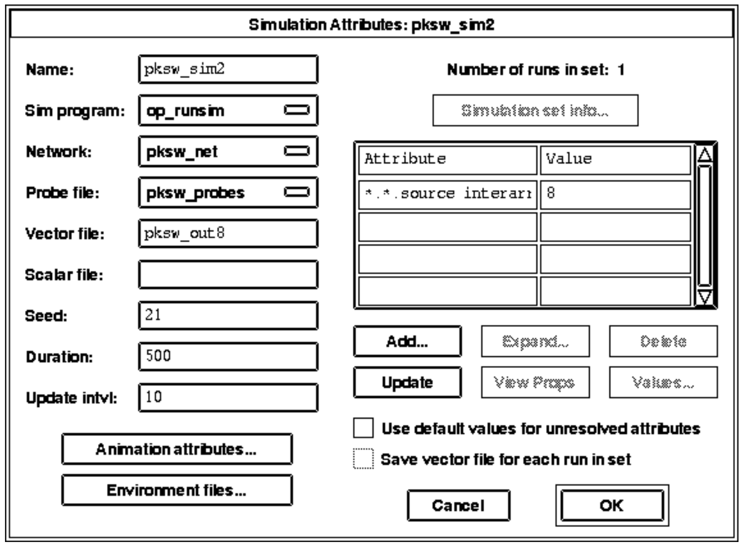
4) Open the attribute dialog box for **`pksw_sim2`** and make the following changes:

a) Change the **`Vector file`** to “**`pksw_out8`**”

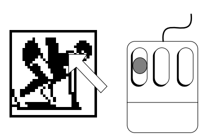
b) Change the **`Attribute`** to “**`*.*.source.interarrival args`**”.

c) Change the **`Value`** to “**`8`**”.

➡ The modified simulation attribute dialog box will look like this:



5) Activate the **`Execute Simulation Sequence`** action button.

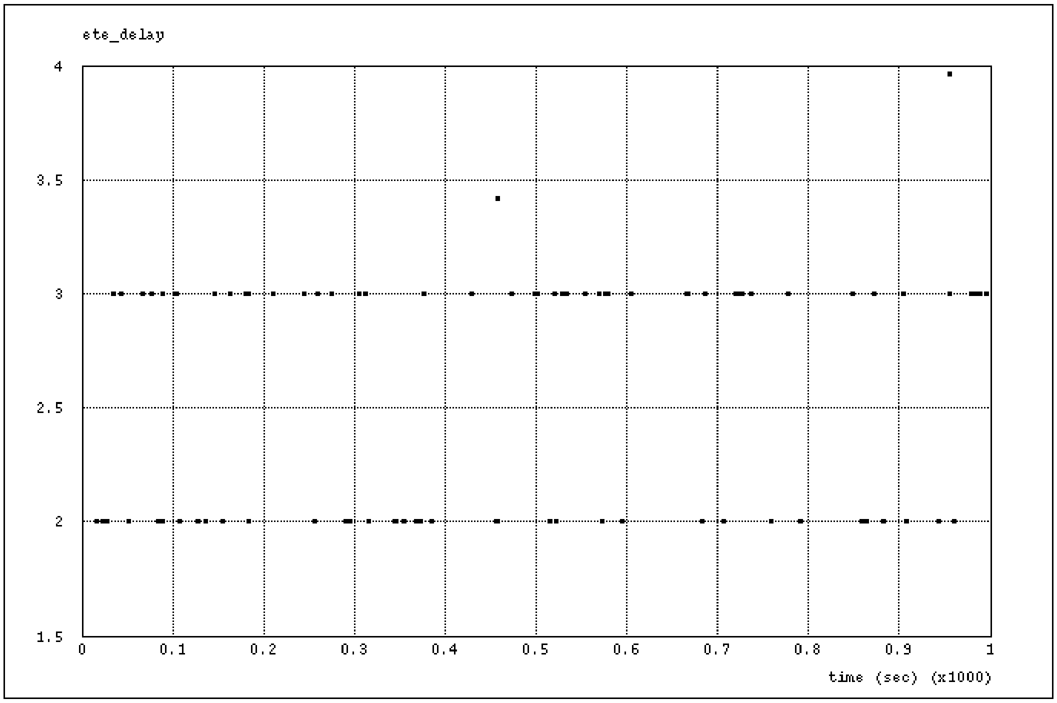


➡ **opnet** displays messages to show when the simulations are complete.

> If the simulation does not execute, refer to the *Troubleshooting Chapter.*

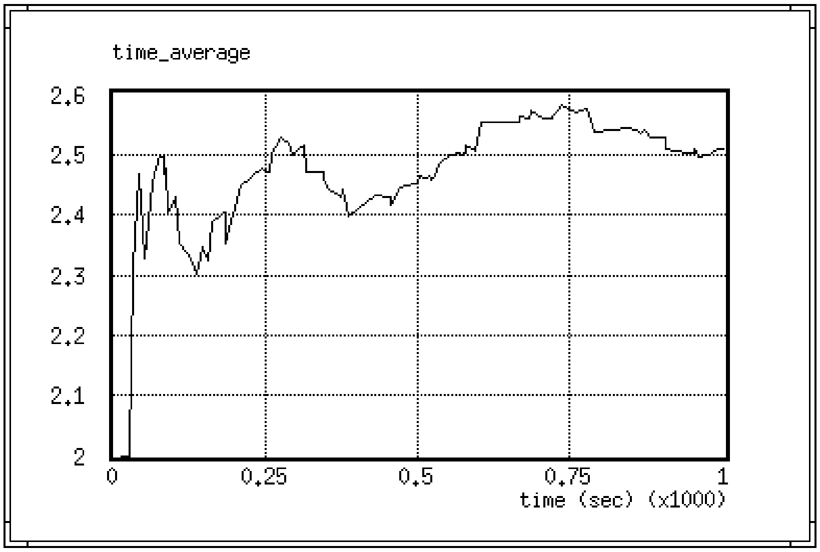
Now you can create graphs with the Analysis Tool similar to those created for the original model (see *Analyzing the Results* on page 182). Looking at a discrete graph of end-to-end delays from the run with **80 second** interarrival times reveals that there are now **two** predominant delay times. (For clarity, the lower ordinate bound has been adjusted in the following diagram with the panel’s **`Edit Panel`** dialog box).

Discrete ete_delay Graph from pksw_out80

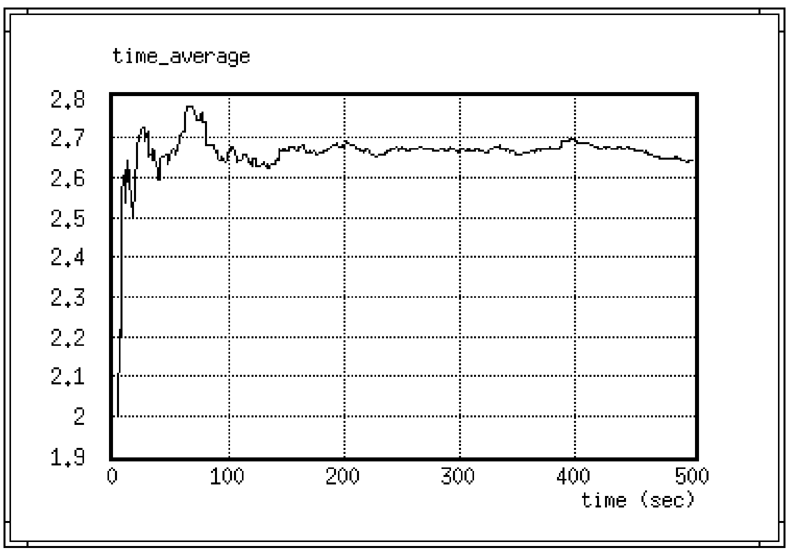


The longer delay times result from those packets addressed to the remote subnet. Looking at a time-average graph of the run with **8 second** interarrival times shows that the average delay is somewhat greater than the average delays experienced in the first model. Refer back to the filtering exercises in section *Mqt.4, Analyzing the Results* of *Chapter Mqt* if needed.

Time Average Graph of ete_delay from pksw_out80



Time Average Graph of ete_delay from pksw_out8



When done experimenting, close all active tools and go on to the next chapter.